ENVI Programmer’s Guide
Restricted Rights Notice

The ENVI® and IDL® software programs and the accompanying procedures, functions, and documentation described herein are sold under license agreement. Their use, duplication, and disclosure are subject to the restrictions stated in the license agreement. Research System, Inc., reserves the right to make changes to this document at any time and without notice.

Limitation of Warranty

Research Systems, Inc. makes no warranties, either express or implied, as to any matter not expressly set forth in the license agreement, including without limitation the condition of the software, merchantability, or fitness for any particular purpose.

Research Systems, Inc. shall not be liable for any direct, consequential, or other damages suffered by the Licensee or any others resulting from use of the ENVI and IDL software packages or their documentation.

Permission to Reproduce this Manual

If you are a licensed user of these products, Research Systems, Inc. grants you a limited, nontransferable license to reproduce this particular document provided such copies are for your use only and are not sold or distributed to third parties. All such copies must contain the title page and this notice page in their entirety.

Acknowledgments

ENVI® and IDL® are registered trademarks of Research Systems, Inc., registered in the United States Patent and Trademark Office, for the computer program described herein. Dancing Pixels, Pixel Purity Index, PPI, n-Dimensional Visualizer, Spectral Analyst, Spectral Feature Fitting, SFF, Mixture-Tuned Matched Filtering, MTMF, 3D-SurfaceView, Band Math, Spectral Math, ENVI Extension, Empirical Flat Field Optimal Reflectance Transformation (EFFORT), and Virtual Mosaic are trademarks of Research Systems, Inc.

Numerical Recipes™ is a trademark of Numerical Recipes Software. Numerical Recipes routines are used by permission.

GRG2™ is a trademark of Windward Technologies, Inc. The GRG2 software for nonlinear optimization is used by permission.

NCSA Hierarchical Data Format (HDF) Software Library and Utilities
Copyright © 1988-1998 The Board of Trustees of the University of Illinois All rights reserved.

CDF Library
Copyright © 1999
National Space Science Data Center
NASA/Goddard Space Flight Center

NetCDF Library
Copyright © 1993-1996 University Corporation for Atmospheric Research/Unidata

HDF EOS Library
Copyright © 1996 Hughes and Applied Research Corporation

This software is based in part on the work of the Independent JPEG Group.

This product contains StoneTable™, by StoneTablet Publishing. All rights to StoneTable™ and its documentation are retained by StoneTablet Publishing, PO Box 12665, Portland OR 97212-0665. Copyright © 1992-1997 StoneTablet Publishing

WASTE text engine © 1993-1996 Marco Piovanelli

Portions of this software are copyrighted by INTERSOLV, Inc., 1991-1998.

Use of this software for providing LZW capability for any purpose is not authorized unless user first enters into a license agreement with Unisys under U.S. Patent No. 4,558,302 and foreign counterparts. For information concerning licensing, please contact: Unisys Corporation, Welch Licensing Department - C1SW19, Township Line & Union Meeting Roads, P.O. Box 500, Blue Bell, PA 19424.

Portions of this computer program are copyright © 1995-1999 LizardTech, Inc. All rights reserved. MsSID is protected by U.S. Patent No. 5,710,835. Foreign Patents Pending.

Other trademarks and registered trademarks are the property of the respective trademark holders.
## Contents

**Preface** .......................................................... 15

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to ENVI</td>
<td>16</td>
</tr>
<tr>
<td>What is ENVI?</td>
<td>16</td>
</tr>
<tr>
<td>Platforms Supported by ENVI</td>
<td>16</td>
</tr>
<tr>
<td>Advantages of ENVI</td>
<td>16</td>
</tr>
<tr>
<td>ENVI, ENVI RT, and IDL</td>
<td>18</td>
</tr>
<tr>
<td>General Review of ENVI Functionality</td>
<td>19</td>
</tr>
<tr>
<td>ENVI Documentation</td>
<td>22</td>
</tr>
<tr>
<td>ENVI Release Notes</td>
<td>22</td>
</tr>
<tr>
<td>ENVI What's New</td>
<td>22</td>
</tr>
<tr>
<td>Installation Guide</td>
<td>22</td>
</tr>
<tr>
<td>ENVI User’s Guide</td>
<td>22</td>
</tr>
<tr>
<td>ENVI Programmer’s Guide</td>
<td>23</td>
</tr>
<tr>
<td>ENVI Tutorials</td>
<td>23</td>
</tr>
</tbody>
</table>

| Reporting Problems | 24 |

---

**ENVI Programmer’s Guide** 3
Chapter 1: Overview ........................................................................................................... 27
Extending ENVI - What is Possible .................................................................................. 28
  Band and Spectral Math Functions ............................................................................... 28
  User Functions ............................................................................................................. 29
  Interactive User Routines ............................................................................................ 29
  Custom File Input ........................................................................................................ 29
  Batch Processing ......................................................................................................... 30
  ENVI Menu Files ......................................................................................................... 30
  Compiling ..................................................................................................................... 30
  Toggle Catch ................................................................................................................. 31
Introduction to ENVI Programming ..................................................................................... 32
  Controlling Complicated Routines with no User Interaction .................................... 32
  Differences in File I/O Between ENVI and IDL ....................................................... 32
  The ENVI and IDL Library Directories .................................................................... 33
  Common Keywords for ENVI Processing Routines ................................................. 34
Opening and Managing Files .............................................................................................. 36
  ENVI_PICKFILE ...................................................................................................... 36
  ENVI_SELECT ......................................................................................................... 36
  ENVI_OPEN_FILE ................................................................................................... 36
  ENVI_FILE_MNG ................................................................................................. 36
  ENVI_GET_FILE_IDS ............................................................................................. 37
Opening External File Types ................................................................................................. 37
Accessing Image Data ....................................................................................................... 38
  ENVI_GET_DATA ................................................................................................... 38
  ENVI_GET_SLICE ................................................................................................... 38
Spatial Subsetting Using ROIs ............................................................................................ 38
Image Output: Making ENVI Format Files ....................................................................... 39
  Saving Image Data to Memory .................................................................................. 39
  Saving Image Data to Disk ......................................................................................... 39
  Creating New Files from Existing ENVI Files ......................................................... 40

Chapter 2: Band and Spectral Math Functions ...................................................................... 41
Band Math Basics .............................................................................................................. 43
  Writing Band Math Functions ....................................................................................... 44
Chapter 3: Batch Processing ................................................................. 51
About Batch Processing ........................................................................... 52
Batch Mode ENVI .................................................................................... 53
Hybrid Batch Mode ................................................................................ 53
Initiating Batch Mode ............................................................................. 55
Exiting Batch Mode ................................................................................ 56
Writing Batch Mode Programs ............................................................... 57
Using ENVI Routines in IDL Programs .................................................... 57
Example: Using COMPILER_OPT .......................................................... 58
Using ENVI Recording to Write Batch Code ........................................ 59
ENVI Message Logging in Batch Mode .................................................. 61
Using the Batch Mode Log File ............................................................... 61
Helpful Tips for Batch Processing ......................................................... 62
Making a Shortcut for Initiating Batch Mode ........................................ 62
Examples of ENVI Batch Programming ................................................ 65
Example: File Statistics (Non-interactive) .............................................. 65
Example: Saturation Stretch (Non-interactive) ...................................... 66

Chapter 4: User Functions ................................................................. 71
User Functions ........................................................................................ 73
Modifying the ENVI Menus ................................................................. 75
Working with the Menu Files ............................................................... 75
Example: Making a Simple User Function .......................................... 77
Adding Widgets to User Functions ....................................................... 82
An Illustrated Glossary of ENVI Widgets ........................................... 83
ENVI_PICKFILE ....................................................................................... 83
ENVI_SELECT .......................................................................................... 83
WIDGET_EDIT ........................................................................................ 84
WIDGET_GEO .............................................................. 85
WIDGET_MAP ............................................................ 85
WIDGET_MENU ........................................................... 86
WIDGET_MULTI ........................................................... 86
WIDGET_OUTF ............................................................ 87
WIDGET_OUTFM ........................................................ 87
WIDGET_PARAM ........................................................ 88
WIDGET_PMENU ........................................................ 88
WIDGET_RGB .............................................................. 89
WIDGET_SLABEL ......................................................... 89
WIDGET_SLIST ............................................................ 90
WIDGET_SSLIDER ........................................................ 90
WIDGET_STRING ........................................................ 91
WIDGET_SUBSET ......................................................... 91
WIDGET_TOGGLE ........................................................ 91
Auto-Managed Widget Events ........................................... 92
  WIDGET_AUTO_BASE ..................................................... 92
  AUTO_WID_MNG ....................................................... 92
Trapping Errors in User Functions ..................................... 98
  Input/Output Error Handling ......................................... 98
  Using CATCH for Unexpected Non-Input/Output Errors ....... 100
Working With Image Displays .......................................... 101
  ENVI_DISP_QUERY ..................................................... 101
  ENVI_GET_IMAGE ...................................................... 101
  DISP_GET_LOCATION .................................................. 101
  DISP_GOTO .............................................................. 101
Using Processing Routines and Tiling ............................... 102
  Tiled Processing Routines ........................................... 102
  Non-Tiled Processing Routines ....................................... 115
  Processing Status Report ............................................. 117
Adapting User Functions for ENVI RT .............................. 120
  Using FORWARD_FUNCTION or COMPILE_OPT STRICTARR .......... 120
  Using RESOLVE_ALL to Find and Compile Dependent Routines .... 120
  Making a Save File ..................................................... 121
Chapter 5: Programming Tools ................................................................. 123

Plotting ........................................................................................................ 125
  Example: Plotting Data ........................................................................... 126
Reports .......................................................................................................... 127
  Example: Creating a Report .................................................................. 127
RGB Colors Triplets .................................................................................. 128
  Example: Getting RGB Color Values .................................................... 128
File Information .......................................................................................... 129
  Example: Basic Image Information ....................................................... 129
  Example: Map Information .................................................................. 130
Managing Files ............................................................................................ 131
  Example: Choosing Files Interactively ................................................ 132

Chapter 6: Interactive User Routines ....................................................... 133

Plot Function ............................................................................................. 135
  Example: Plot Function ....................................................................... 137
Spectral Analyst Functions ........................................................................ 138
  Example: Spectral Analyst Function ..................................................... 140
User Defined Map Projection Types ........................................................ 142
  Example: User Defined Map Projection ............................................... 144
User Defined Units ..................................................................................... 147
User Move Routines ................................................................................... 148
  Example: Simple User Defined Move Routine ..................................... 149
  Example: Widget User Defined Move Routine .................................... 150

Chapter 7: Custom File Input ................................................................. 153

Types of Image Ingest ................................................................................ 154
Parsing Image File Headers ...................................................................... 155
  Example: Parsing a Keyword/Value Header ........................................ 155
  Example: Parsing a Positional Header ................................................ 156
Custom Image Read Procedures ............................................................. 158
Spatial Read Procedure ............................................................................. 160
  Example: Unsigned Integer Spatial Reader ......................................... 161
Spectral Read Procedure .......................................................................... 163
Example: Unsigned Integer Spectral Reader ........................................................... 164

Chapter 8:
Additional Topics in ENVI Programming ............................................................... 167
Coordinate Systems in ENVI ..................................................................................... 168
   Image Coordinates (or Pixel Coordinates) ............................................................ 168
   File Coordinates ................................................................................................... 168
   XSTART and YSTART ......................................................................................... 168
Regions of Interest ..................................................................................................... 171
   Processing with ROIs ............................................................................................ 171
   Selecting ROIs ........................................................................................................ 172
   Using ROI Data ...................................................................................................... 176
   Using ROI DIMS Pointers ...................................................................................... 178
   Using ROI Addresses ............................................................................................ 180
Using the Endmember Collection Widget ................................................................. 182
Programmatically Controlling ENVI Image Display Windows .................................. 185
   Getting the Window ID for Scroll, Image and Zoom Windows ............................ 185
   Getting the Zoom factor of the Zoom Window and the Resize Factor of the Scroll Window ........................................................................................................ 185
   Getting the Position of the Current Pixel ............................................................. 185
   Moving the Zoom Window to a New Pixel Location ............................................ 185
   Closing an ENVI Image Display Group ................................................................. 185
ENVI Installation Directory Components .................................................................... 187
   ENVI Subdirectories .............................................................................................. 187
   The Menu Directory .............................................................................................. 188
   The Map_Proj Directory ....................................................................................... 189
Basic ENVI Library Routines Reference ..................................................................... 190
Quick Reference of ENVI Library Function Variable Codes ....................................... 196

Chapter 9:
ENVI Routines ......................................................................................................... 201
How to Use this Chapter ............................................................................................ 202
Available Routines ..................................................................................................... 205
   ADAPT_FILT_DOIT ............................................................................................... 217
   AIRSAR_HEADER_DOIT .................................................................................... 221
   AIRSAR_PED_HEIGHT_DOIT ........................................................................... 223
   AIRSAR_PHASE_IMAGE_DOIT .......................................................................... 226
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIRSAR_POLSIG_DOIT</td>
<td>229</td>
</tr>
<tr>
<td>AIRSAR_SCATTER_DOIT</td>
<td>232</td>
</tr>
<tr>
<td>AIRSAR_SYNTH_DOIT</td>
<td>236</td>
</tr>
<tr>
<td>ASPECT_DOIT</td>
<td>240</td>
</tr>
<tr>
<td>AUTO_WID_MNG</td>
<td>243</td>
</tr>
<tr>
<td>BAD_DATA_DOIT</td>
<td>244</td>
</tr>
<tr>
<td>CATCH</td>
<td>247</td>
</tr>
<tr>
<td>CF_DOIT</td>
<td>248</td>
</tr>
<tr>
<td>CLASS_CONFUSION_DOIT</td>
<td>251</td>
</tr>
<tr>
<td>CLASS_CS_DOIT</td>
<td>256</td>
</tr>
<tr>
<td>CLASS_DOIT</td>
<td>260</td>
</tr>
<tr>
<td>CLASS_MAJORITY_DOIT</td>
<td>270</td>
</tr>
<tr>
<td>CLASS_RULE_DOIT</td>
<td>273</td>
</tr>
<tr>
<td>CLASS_STATS_DOIT</td>
<td>276</td>
</tr>
<tr>
<td>COM_CLASS_DOIT</td>
<td>280</td>
</tr>
<tr>
<td>CONTINUUM_REMOVE_DOIT</td>
<td>283</td>
</tr>
<tr>
<td>CONV_DOIT</td>
<td>286</td>
</tr>
<tr>
<td>CONVERT_DOIT</td>
<td>289</td>
</tr>
<tr>
<td>CONVERT_INPLACE_DOIT</td>
<td>291</td>
</tr>
<tr>
<td>CROSS_TRACK_CORRECTION_DOIT</td>
<td>293</td>
</tr>
<tr>
<td>DARK_SUB_DOIT</td>
<td>297</td>
</tr>
<tr>
<td>DECOR_DOIT</td>
<td>300</td>
</tr>
<tr>
<td>DEM_BAD_DATA_DOIT</td>
<td>303</td>
</tr>
<tr>
<td>DESKEW_DOIT</td>
<td>306</td>
</tr>
<tr>
<td>DESTRIPE_DOIT</td>
<td>309</td>
</tr>
<tr>
<td>DISP_GET_LOCATION</td>
<td>312</td>
</tr>
<tr>
<td>DISP_GOTO</td>
<td>314</td>
</tr>
<tr>
<td>DISP_OUT_IMG</td>
<td>317</td>
</tr>
<tr>
<td>ELINE_CAL_DOIT</td>
<td>320</td>
</tr>
<tr>
<td>EMITTANCE_CALC_DOIT</td>
<td>323</td>
</tr>
<tr>
<td>ENVI</td>
<td>326</td>
</tr>
<tr>
<td>ENVI_ADD_PROJECTION</td>
<td>327</td>
</tr>
<tr>
<td>ENVI_AVHRR_CALIBRATE_DOIT</td>
<td>328</td>
</tr>
<tr>
<td>ENVI_AVHRR_GEOMETRY_DOIT</td>
<td>331</td>
</tr>
<tr>
<td>ENVI_AVHRR_WARP_DOIT</td>
<td>334</td>
</tr>
<tr>
<td>ENVI_BATCH_EXIT</td>
<td>339</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>ENVI_Batch_Init</td>
<td>341</td>
</tr>
<tr>
<td>ENVI_Batch_Status_Window</td>
<td>342</td>
</tr>
<tr>
<td>ENVI_Buffer_Zone_DoIt</td>
<td>345</td>
</tr>
<tr>
<td>ENVI_Cal_DoIt</td>
<td>348</td>
</tr>
<tr>
<td>ENVI_Center</td>
<td>351</td>
</tr>
<tr>
<td>ENVI_Clover_DoIt</td>
<td>352</td>
</tr>
<tr>
<td>ENVI_Collect_Spectra</td>
<td>355</td>
</tr>
<tr>
<td>ENVI_Compute_Sun_Angles</td>
<td>358</td>
</tr>
<tr>
<td>ENVI_Convert_File_Coordinates</td>
<td>360</td>
</tr>
<tr>
<td>ENVI_Convert_File_Map_Projection</td>
<td>362</td>
</tr>
<tr>
<td>ENVI_Convert_Projection_Coordinates</td>
<td>366</td>
</tr>
<tr>
<td>ENVI_Create_ROI</td>
<td>369</td>
</tr>
<tr>
<td>ENVI_Cube_3D_DoIt</td>
<td>370</td>
</tr>
<tr>
<td>ENVI_Default_Sretch_Creat</td>
<td>373</td>
</tr>
<tr>
<td>ENVI_Define_ROI</td>
<td>376</td>
</tr>
<tr>
<td>ENVI_Delete_ROIs</td>
<td>379</td>
</tr>
<tr>
<td>ENVI_Disp_Query</td>
<td>380</td>
</tr>
<tr>
<td>ENVI_Display_Bands</td>
<td>383</td>
</tr>
<tr>
<td>ENVI_DoIt</td>
<td>386</td>
</tr>
<tr>
<td>ENVI_Enter_Data</td>
<td>389</td>
</tr>
<tr>
<td>ENVI_EVF_Close</td>
<td>396</td>
</tr>
<tr>
<td>ENVI_EVF_Define_Add_Record</td>
<td>398</td>
</tr>
<tr>
<td>ENVI_EVF_Define_Close</td>
<td>400</td>
</tr>
<tr>
<td>ENVI_EVF_Define_Init</td>
<td>402</td>
</tr>
<tr>
<td>ENVI_EVF_Info</td>
<td>404</td>
</tr>
<tr>
<td>ENVI_EVF_Open</td>
<td>406</td>
</tr>
<tr>
<td>ENVI_EVF_Read_Record</td>
<td>408</td>
</tr>
<tr>
<td>ENVI_File_Mng</td>
<td>410</td>
</tr>
<tr>
<td>ENVI_File_Query</td>
<td>411</td>
</tr>
<tr>
<td>ENVI_File_Type</td>
<td>417</td>
</tr>
<tr>
<td>ENVI_Filter_DoIt</td>
<td>420</td>
</tr>
<tr>
<td>ENVI_Get_Configuration_Values</td>
<td>422</td>
</tr>
<tr>
<td>ENVI_Get_Data</td>
<td>423</td>
</tr>
<tr>
<td>ENVI_Get_Display_Numbers</td>
<td>426</td>
</tr>
<tr>
<td>ENVI_Get_File_Ids</td>
<td>427</td>
</tr>
<tr>
<td>ENVI_Get_Image</td>
<td>428</td>
</tr>
</tbody>
</table>
ENVI_GET_MAP_INFO .............................................................................................. 430
ENVI_GET_PROJECTION .......................................................................................... 431
ENVI_GET_RGB_TRIPLETS ..................................................................................... 433
ENVI_GET_ROI ........................................................................................................... 435
ENVI_GET_ROI_DATA .............................................................................................. 436
ENVI_GET_ROI_DIMS_PTR ..................................................................................... 438
ENVI_GET_ROI_IDS .................................................................................................. 439
ENVI_GET_SLICE ....................................................................................................... 441
ENVI_GET_STATISTICS .......................................................................................... 443
ENVI_GET_TILE ......................................................................................................... 446
ENVI_GRID_DOIT ...................................................................................................... 448
ENVI_INFO_WID ........................................................................................................ 451
ENVI_INIT_TILE ......................................................................................................... 452
ENVI_IO_ERROR ........................................................................................................ 454
ENVI_LAYER_STACKING_DOIT ............................................................................ 455
ENVI_MAP_INFO_CREATE ...................................................................................... 460
ENVI_MASK_APPLY_DOIT ...................................................................................... 465
ENVI_NEURAL_NET_DOIT ...................................................................................... 468
ENVI_OPEN_DATA_FILE ......................................................................................... 473
ENVI_OPEN_FILE ....................................................................................................... 476
ENVI_OUTPUT_TO_EXTERNAL_FORMAT ........................................................... 477
ENVI_PICKFILE .......................................................................................................... 479
ENVI_PLOT_DATA .................................................................................................... 481
ENVI_PROJ_CREATE ................................................................................................. 483
ENVI_READ_COLS .................................................................................................... 487
ENVI_REPORT_INC ................................................................................................... 488
ENVI_REPORT_INIT .................................................................................................. 489
ENVI_REPORT_STAT ............................................................................................... 491
ENVI_RESTORE_ROIS .............................................................................................. 493
ENVI_ROI_TO_IMAGE_DOIT ................................................................................... 494
ENVI_SAVE_ROIS ...................................................................................................... 496
ENVI_SEGMENT_DOIT ............................................................................................. 497
ENVI_SELECT ............................................................................................................. 500
ENVI_SENSOR_TYPE ................................................................................................ 504
ENVI_SET_INHERITANCE ....................................................................................... 507
ENVI_SETUP_HEAD .................................................................................................. 510
ENVI_STATS_DOIT .................................................................................................... 519
ENVI_SUM_DATA_DOIT ........................................................................................... 525
ENVI_SYNTHETIC_COLOR_DOIT ........................................................................... 529
ENVI_TILE_DONE ...................................................................................................... 532
ENVI_TOGGLE_CATCH ............................................................................................ 533
ENVI_TRANSLATE_PROJECTION_NAME .................................................................. 534
ENVI_TRANSLATE_PROJECTION_UNITS ............................................................. 535
ENVI_USER_DEFINED_ANNOTATION ................................................................. 536
ENVI_WRITE_DBF_FILE ........................................................................................... 543
ENVI_WRITE_STATISTICS ....................................................................................... 545
FFT_DOIT ..................................................................................................................... 548
FFT_INV_DOIT ............................................................................................................ 551
GAINOFF_DOIT ........................................................................................................... 554
GEN_IMAGE_DOIT ..................................................................................................... 557
HANDLE_VALUE ........................................................................................................ 560
HIST_EXPORT_DOIT .................................................................................................. 562
MAGIC_MEM_CHECK ............................................................................................... 566
MATCH_FILTER_DOIT .............................................................................................. 568
MATCH_FILTER_MT_DOIT ...................................................................................... 572
MATH_DOIT ................................................................................................................ 576
MNF_DOIT ................................................................................................................... 579
MNF_INV_DOIT ........................................................................................................... 583
MORPH_DOIT .............................................................................................................. 585
MOSAIC_DOIT ............................................................................................................ 588
MSSCAL_DOIT .......................................................................................................... 594
MUNSELL_DOIT ......................................................................................................... 598
MUNSELL_INV_DOIT ................................................................................................ 600
NDVI_DOIT .................................................................................................................. 602
ON_IOERROR .............................................................................................................. 605
PC_ROTATE ................................................................................................................. 606
PPI_DOIT ...................................................................................................................... 610
RADAR_INC_ANGLE_DOIT ...................................................................................... 613
RATIO_DOIT ................................................................................................................ 616
REG_WARP_DOIT ....................................................................................................... 619
RESIZE_DOIT .............................................................................................................. 624
RGB_GET_BANDS ..................................................................................................... 627
Preface

This chapter covers the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to ENVI</td>
<td>16</td>
</tr>
<tr>
<td>ENVI, ENVI RT, and IDL</td>
<td>18</td>
</tr>
<tr>
<td>General Review of ENVI Functionality</td>
<td>19</td>
</tr>
<tr>
<td>ENVI Documentation</td>
<td>22</td>
</tr>
<tr>
<td>Reporting Problems</td>
<td>24</td>
</tr>
</tbody>
</table>
Introduction to ENVI

What is ENVI?

ENVI (the Environment for Visualizing Images) is a revolutionary image processing system. From its inception, ENVI was designed to address the numerous and specific needs of those who regularly use satellite and aircraft remote sensing data. ENVI provides comprehensive data visualization and analysis for images of any size and any type – all from within an innovative and user-friendly environment.

Platforms Supported by ENVI

ENVI runs on the following platforms (for more details see the ENVI Installation Guide):

- multiple UNIX platforms and LINUX
- Windows 98, Windows NT and Windows 2000 on 80386 (and higher) processors

Advantages of ENVI

ENVI’s developers are scientists who actively participate in remote sensing research. Because of their interests in the development of the field, these scientists continually supplement ENVI with leading-edge technology and innovative and useful functions. ENVI’s power and applicability to current remote sensing problems are continually expanding.

One of ENVI’s strengths lies in its unique approach to image processing - it combines file-based and band-based techniques with interactive functions. Once a data input file is opened, each band is stored in a list so that it can be manipulated by all system functions. If you open multiple files at once, you can select bands of disparate data types to be processed as a group. ENVI displays these bands in 8- or 24-bit displays; all windows are resizeable. ENVI’s display window groups consist of a main image window, a zoom window, and a scroll window. ENVI provides its users with many unique interactive analysis capabilities, accessed from within these windows. ENVI’s multiple dynamic overlay capabilities allow easy comparison of images in multiple displays. Real-time extraction and linked spatial/spectral profiling from multiband and hyperspectral data give users new ways of looking at high-dimensional data. ENVI also provides interactive tools to view and analyze vectors and GIS attributes. Standard capabilities such as contrast stretching and 2-dimensional scatter plots are just a few of the interactive functions available to ENVI users.
ENVI’s strong visual interface is complemented by its comprehensive library of processing algorithms. ENVI includes all the basic image processing functions with a friendly, interactive point-and-click graphical user interface (GUI). Because of the ease of working with ENVI’s GUI, many basic processing functions become easier to utilize. Some of these functions include data transforms, filtering, classification, registration and geometric corrections, spectral analysis tools, and radar tools. ENVI does not impose limitations on the number of spectral bands you can process; therefore, you can use ENVI’s tools on multi-spectral or hyperspectral data sets. ENVI also includes many advanced functions that allow for analysis of radar data sets.

ENVI addresses common image processing problem areas, such as input of non-standard data types, viewing and analysis of large images, and simple extension of analysis capabilities by users (plug-in functions). The software includes essential tools required for image processing across multiple disciplines, and has the flexibility to allow you to implement your own analysis strategies.
ENVI, ENVI RT, and IDL

ENVI is written in Interactive Data Language (IDL), a powerful structured programming language that offers integrated image processing. IDL is required to run ENVI and the flexibility of ENVI is due largely to IDL’s capabilities. There are two types of ENVI licenses, ENVI with a full version of IDL, and ENVI RT which includes a runtime version of IDL. ENVI users can utilize IDL’s strengths by customizing their own command-line functions. Advanced ENVI users should find the flexibility offered by IDL’s interactive features helpful for their dynamic image analyses. ENVI RT provides complete ENVI functionality, but custom routines cannot be written.

More information on IDL can be found in the IDL User’s Guide, IDL Reference Guide, and other IDL documentation.
General Review of ENVI Functionality

ENVI simplifies comprehensive interactive processing of large multiband data sets, screen-sized images, spectral plots and libraries, and image regions-of-interest, all while providing flexible display capabilities and geographic-based image browsing. ENVI provides a multitude of interactive functions, including X, Y, Z profiling; image transects; linear and non-linear histogramming and contrast stretching; color tables, density slicing and classification color mapping; quick filter preview; and Region of Interest definition and processing. Simple methods are provided for locating specific pixels, and for interactive spatial/spectral pixel editing. ENVI also offers interactive scatter plot functions, including 2-D “Dancing Pixels” and n-Dimensional Visualization. With ENVI, you are able to link images and create “Dynamic Overlays” interactively. You also have the power to create comprehensive vector overlays with GIS attributes, add map and pixel grids, and annotate your images. Other ENVI interactive functions include 3-D (perspective) viewing, surface shading, image draping, image animation (“movies”), and geometric rectification and mosaicking.

ENVI’s strengths are not limited to its interactive mode: ENVI functions work with full data files and subsets. We provide you with a complete set of tools to process panchromatic images, AVHRR, Landsat MSS data, Landsat TM data, and other multispectral and hyperspectral images as well as data from today’s and tomorrow’s advanced SAR systems. AVHRR tools allow display of ephemeris data, data calibration, geometric rectification, and calculation of sea-surface temperature. Landsat tools include functions for de-skewing and correcting aspect ratios for pre-1979 MSS data, destriping and correcting for atmospheric interference, and calibrating to reflectance using pre-launch gains and offsets. Tools for calibrating thermal infrared data to emissivity using three methods are included in ENVI.

General-purpose image processing functions include data transformations such as principal components transformations, band ratios transformations, hue-saturation-value (HSV), decorrelation stretching, and generation of vegetation indices. Filtering functions include convolution kernels for low pass, high pass, median, directional, and edge detection. ENVI supports user-defined convolution kernels up to M x N; and all kernels can be interactively edited. Other specific filter types such as Sobel, Roberts, dilation, and erosion are also provided, along with adaptive filters such as Lee, Frost, Gamma, and Kuan. Texture filters, including measures for data range, mean, variance, entropy, skewness, variance, homogeneity, contrast, dissimilarity, entropy, and correlation are provided.
With ENVI, you can also easily perform frequency-domain filtering (FFT filtering) functions, like forward FFT transformations, interactive frequency domain filtering, and inverse FFT computations.

ENVI provides KMEANS and Isodata unsupervised classification methods. Supervised classification in ENVI uses standardized training-set collection methodologies to simplify parallelepiped, minimum distance, maximum likelihood, and mahalanobis distance classifiers. ENVI also provides the Binary Encoding and Spectral Angle Mapper (SAM) spectral classifiers. Post-classification processing includes clump, sieve, combine classes, and an interactive classification display tool, which allow generalization of image-maps prior to export to GIS vector files. ENVI’s confusion matrix/Kappa Coefficient images and statistics allow assessment of classification accuracy. Generation of ROC Curves can be used to visualize a classifier’s performance in order to select a decision threshold.

ENVI provides a full suite of tools for processing hyperspectral data, including special mapping tools for linear spectral unmixing and matched filtering using either image or library endmembers. You can use ENVI routines to access spectral libraries and compare library spectra to image spectra. The Pixel Purity Index (PPI) allows users to find the most spectrally pure pixels in an image to determine spectral endmembers. The unique n-Dimensional visualizer in ENVI performs interactive animation of scatterplots in n-dimensions, providing a dynamic view of the data that allows determinative selection of endmember materials and their corresponding spectra. Spectral Feature Fitting of hyperspectral data to library spectra aids you in identifying rocks, minerals, vegetation, and other materials. The linear spectral unmixing function allows determination of relative spectral abundances using multispectral and hyperspectral data. ENVI also provides the “Spectral Analyst” to identify materials based on comparison to spectral libraries. Flexible “Band Math” and “Spectral Math” functions allow users to enter complex mathematical expressions, functions, and procedures that access IDL’s powerful array-processing functions.

ENVI also contains comprehensive SAR processing capabilities. All standard ENVI processing routines are inherently SAR-capable. In addition, a full suite of SAR-specific analysis methods is provided, including ingest of standard SAR formats such as Radarsat and ERS-1; review and reading of header information from CEOS-format data; antenna pattern correction, slant-to-ground range correction and generation of incidence angle images. Other SAR functions include adaptive and texture filters, creating a synthetic color image, and a broad range of polarimetric data analysis methods.
Finally, ENVI provides tools to take image data to final map form. These include both image-to-image and image-to-map registration, basic orthorectification, image mosaicking, and map composition utilities. An integrated set of vector GIS import, export, and analysis tools provide the capability to bring industry-standard GIS data into ENVI, view and analyze vectors and GIS attributes (as image overlays as well as in a separate vector analysis window), modify existing vectors, query attributes, utilize vector layers for raster analysis or generate new vector GIS layers from raster image processing results, and create standard GIS export-format files.

ENVI provides all of these capabilities and more in one tightly integrated image analysis package. For additional information about specific functionality, please see the corresponding sections of interest in this programmer’s guide or in the ENVI User’s Guide.
ENVI Documentation

ENVI’s documentation set includes Release Notes (distributed as a .txt file on the ENVI CD and installed into the ENVI installation directory), the *What’s New in ENVI* guide, the *ENVI User’s Guide*, the *ENVI Programmer’s Guide* and the *ENVI Tutorials*. All documentation (except the Release Notes) is distributed with ENVI in Adobe Acrobat PDF format.

**Note**

To view the ENVI manuals in PDF format, you need Adobe Acrobat 3.0 or later. Acrobat Reader 4.0 with Search is included on your product CD-ROM. Visit Adobe Systems Web site at [www.adobe.com](http://www.adobe.com) for the latest information and downloads concerning Acrobat Reader.

ENVI Release Notes

The Release Notes are included on the ENVI CD and contain information about new functions, changes, updates, and issues related to the new release.

ENVI What’s New

The *What’s New in ENVI* guide provides a synopsis of all new functions and changes in the new release. Detailed instructions are included in the *ENVI User’s Guide* and *ENVI Programmer’s Guide*.

Installation Guide

Describes how to install and license ENVI.

ENVI User’s Guide

The *ENVI User’s Guide* provides step-by-step instructions on how to use ENVI’s graphical user interface, interactive functions, display capabilities, function parameter descriptions, and customizing options. All general information about ENVI and its functionality is found in this guide.
ENVI Programmer’s Guide

The ENVI Programmer’s Guide provides instructions on programming in ENVI. Common ENVI extensions include band and spectral math functions, user functions, custom spatial, spectral or region of interest (ROI) processing, custom file input routines, batch processing, and other report and plotting tools. The suite of ENVI routines available to the programmer greatly simplifies development of these routines while maintaining the same “look-and-feel.”

ENVI Tutorials

ENVI includes tutorials designed to lead the new user through the program’s basic functionality. These include some remote sensing background and step-by-step procedures to execute the corresponding ENVI functions. Some of the tutorials provided are: multispectral classification, georeferencing and registration, mosaicking, vector overlay and GIS analysis, map composition, several hyperspectral data analysis tutorials, and both basic and advanced SAR processing. Sample datasets are provided on the ENVI Data CD for use with these tutorials.
Reporting Problems

When you encounter a problem with ENVI, please report them as quickly as possible so we can fix it. This section is intended to help you report problems in a way that will help us correct the problem rapidly.

Background Information

When a bug is reported and verified, we correct it in a later release. Sometimes, a bug only occurs when running on a certain machine, operating system, or graphics device. For these reasons, we need to know the following facts when you report a bug:

- Your ENVI installation number.
- The version of ENVI you are running.
- The type of machine it is running on.
- The operating system version it is running under.
- The type and version of your windowing system.
- The graphics device, if the problem involves graphics.

The installation number is assigned by us when you purchase ENVI. The ENVI version, site number, and type of machine are printed when ENVI is started.

Under Unix, the version of the operating system can usually be found in the file /etc/motd. It is also printed when the machine boots. In any event, your system administrator should know.


On the Macintosh, select “About this Macintosh” from the apple menu.

Double Check

Before reporting a problem, you should always ask yourself “Is it really a bug?” Sometimes, it is a simple matter of misinterpreting what is supposed to happen. Double check with the manual or a local expert. Make sure your system is properly configured with enough virtual memory and sufficient operating system quotas.

If you cannot determine what should happen in a given situation by consulting the reference manual, the manual needs to be improved on that topic. Please let us know if you feel that the manual was vague or unclear on a subject.
Describing The Problem

When describing the problem, it is important to use precise language. Vague terms like “crashes”, “blows up”, and “fails” are open to many interpretations. Does it really crash ENVI and leave you looking at an operating system prompt? This would be our interpretation of “crash.” Perhaps, however, it just issues an unexpected error message and gives another prompt. What is really meant by a term like “fails?”

Sending Data with Your Bug Report

If the data files are required to reproduce the bug, we will need you to send them to us on magnetic media or via e-mail. Call us for details.

Contact Us

To report a problem, contact us at the following addresses.

Mail

Research Systems, Inc.
4990 Pearl East Circle
Boulder, Colorado 80301

Telephone

(303) 786-9900 (Voice)
(303) 786-9909 (Fax)
(303) 413-3920 (IDL technical support direct line)

Electronic Mail

support@ResearchSystems.com
Chapter 1: Overview

This chapter covers the following topics:

- Extending ENVI - What is Possible ...... 28
- Introduction to ENVI Programming ...... 32
- Opening and Managing Files ............ 36
- Accessing Image Data ................. 38
- Image Output: Making ENVI Format Files 39
Extending ENVI - What is Possible

The term “Extending ENVI” has a broad meaning and covers a variety of customizations. It can range from creating a library of band and spectral math functions to complex processing and graphical interface development. Whether you are creating simple enhancements or large scale complex additions, it is useful to understand the programming concepts and tools used in ENVI.

Common ENVI extensions include band and spectral math functions, custom spatial, spectral, or region of interest (ROI) processing, user functions, custom file input routines, batch processing, and other report and plotting tools. The suite of ENVI routines available to the programmer greatly simplifies development of custom routines while maintaining the same “look-and-feel” as ENVI. This manual provides an array of interactive examples and sample routines to aid in the understanding and development of custom routines.

Note
The ENVI command line and programming extensibility are not available if you have purchased ENVI RT (an ENVI installation that uses the runtime version of IDL).

Band and Spectral Math Functions

One of the simplest interfaces available to the programmer is through Band Math and Spectral Math. Simple functions can be added and ENVI will handle the data input and output and user interfaces. Most band and spectral math expressions can be entered interactively through the Band or Spectral Math expression dialog. However, you may want to write a specialized function for processing either spatial (Band Math) or spectral (Spectral Math) data. Band Math allows you to input data from any bands (or a file), process them, and output a band. Spectral Math allows you to input spectra from a plot or file, process then, and output a spectrum. All of the input and output data selections, accessing the data, calling the function, and outputting the result are handled by the Band or Spectral Math function in ENVI.

Therefore, you do not need to make menu changes, create parameter widgets, or perform I/O as you would need to in a processing routine. You only need to provide the processing calculation within your function. See Chapter 2, “Band and Spectral Math Functions” for more information.
User Functions

User functions can be written in IDL, C, Fortran, or other high level languages and are integrated into the ENVI software and executed from the ENVI menus. User functions can get input data from ENVI and enter results directly into ENVI. ENVI provides a library of procedures and programming tools written in IDL to handle input, output, plotting, reports, and file management. Also, many of the ENVI internal processing functions (such as classification routines) are available for use in user functions or batch routines. A set of ENVI compound widgets are provided to simplify writing widget interfaces and give your routines the same look and feel as ENVI. See Chapter 4, “User Functions”, and Chapter 5, “Programming Tools” for more information.

The user function consists of a widget definition, event handler, and processing routine. The user function is assigned a button on the ENVI menu and is executed like any other ENVI function. The widget definition or user interface is called when the item is selected from the menu and prompts the user for all processing parameter inputs. Associated with the widget may be an event handler to manage widget inputs. ENVI has an excellent method of auto-managing widget events that often eliminates the need for the event handler. The processing routine takes the input image data, processes the data, and outputs a new image, plot, report or other result.

Interactive User Routines

Interactive routines are processes that ENVI applies or calls automatically in a session. Users can supply additional routines to use alongside the default methods in ENVI. Interactive routines can be added for plot functions, spectral analysis functions, and user defined move routines. See Chapter 6, “Interactive User Routines” for more information.

Custom File Input

You can write a custom file input routine to open and read your data format on-the-fly. When opening an unsupported file format automatically (without prompting for the file information) the input routine parses the file header and then places the bands in the Available Bands List. Data stored in unsupported storage formats can then be accessed on-the-fly in ENVI using custom readers. When processing the data, the file will be accessed using your custom reader without converting to an ENVI format. See Chapter 7, “Custom File Input” for more information.
Batch Processing

Batch processing, non-interactive routines that string together ENVI processing functions, can be written using the ENVI_DOIT procedure. ENVI_DOIT provides the processing portion of a user function and is run without any user interaction. Batch processing can be started from a menu event or in ENVI’s non-interactive mode. See Chapter 3, “Batch Processing” for more information.

ENVI Menu Files

The ENVI main menu, envi.men, and display menu, display.men, are configurable items that can be found in the menu subdirectory of the ENVI installation. The envi.men file defines the main ENVI menu and the display.men file defines the Main Image window Display menu. Both menus are defined in an ASCII file which outlines the placement of menu buttons, pulldowns, and separators. The procedure called when the menu item is selected is also defined in the menu file. You can reposition menu items or add new items, depending on your needs and preferences. ENVI does not distinguish between ENVI and user event handlers, which allows for easy integration of user events. See “User Functions” on page 73 for more information.

Compiling

Custom routines (Band and Spectral Math Functions, User Functions, Interactive User Routines, and Custom File Readers) must either be compiled manually or placed in the save_add directory of the ENVI installation where they are auto compiled during ENVI start-up. Alternatively, if the files and the routines they contain are named properly, IDL will automatically find and compile the routines the first time they are used as long as they are in the IDL_PATH. Manually compiling the files using the File > Compile IDL Module is recommended during development to debug the routines. Once the routines are completed and working properly, place them into the save_add directory to allow auto-compiling each time ENVI is started. Please note that you can change the location of the save_add directory, as desired, in your ENVI preferences or configuration file.

Batch routines are compiled using the .run command on the IDL command line.

Note

ENVI RT users cannot compile routines. However, they can place a compiled (.sav) file in the save_add directory which will be automatically restored at start-up time.
Toggle Catch

When developing user functions it may be useful to disable the mechanism ENVI uses to catch errors. Displaying the “catch” mechanism will cause ENVI to halt execution at the error and allow examination of the routines variables. The error message will be printed in the IDL log window and variables can be examined using the ENVI command line.

- To toggle the catch mechanism on and off, type `ENVI_TOGGLE_CATCH` on the ENVI Command Line (see “ENVI_TOGGLE_CATCH” on page 533).
Introduction to ENVI Programming

There are currently more than 150 ENVI processing routines available that encompass virtually all of the functionality provided in the interactive ENVI program. Each processing routine is an IDL procedure or function, and is used just as any other IDL routine. A complete index to these functions and a full reference page for each is located in Chapter 9, “ENVI Routines”.

Note
Remember, you can run the ENVI Online Help even when you’re not in ENVI. On Windows, from the Start Menu select Programs > ENVI 3.5 > ENVI Online Help. On UNIX, just type `envihelp` at the command prompt, and on the Macintosh, double-click the ENVI Help icon in the RSI/ENVI folder.

Controlling Complicated Routines with no User Interaction

Most of ENVI’s processing routines require a great deal of user interaction. When writing your own code to call ENVI processing routines, all aspects of a processing routine must be handled explicitly by the programmer. As a result, most of the ENVI processing routines will require many more keywords than a typical IDL procedure or function might use. In fact, because there is often so much information that must be passed into an ENVI processing routine, these routines typically use only keywords instead of positional parameters, to prevent the user from having to pass information in a specific sequence.

For example, consider performing a simple maximum likelihood classification in your own code. The same classification can be performed in batch mode using the routine ENVI_DOIT with the CLASS_DOIT keyword. When such a classification is done in ENVI interactively, the user must specify the name of the file being classified, spatial and spectral subsets for the input file, ROIs for use as the training sets, whether the ROI data should be collected from an input file or another file, a probability threshold, whether to generate rule images, and whether the results should be saved to file or memory. All of these parameters must now be specified in your code using keywords to ENVI_DOIT.

Differences in File I/O Between ENVI and IDL

File I/O for ENVI programming differs quite markedly from how it is handled in IDL. In IDL, file I/O requires obtaining a Logical Unit Number (or LUN) for the file, and the use of procedures such as OPENR, READU, OPENW, and WRITEU to read from and write to the file. In contrast, all file I/O for the ENVI processing
routines is controlled internally, so the ENVI programmer never needs to obtain an LUN. Instead, all ENVI processing routines require that the input file is specified by a unique File ID, or FID. The FID is essentially a pointer to the data file, but it is not a LUN. When a file needs to be accessed, ENVI internally obtains an LUN for the file, reads or writes the required data, and then frees the LUN. Thus ENVI does not consume or reserve any LUNs. This method of file I/O allows an unlimited number of files to be opened in ENVI simultaneously, even though IDL only provides 128 LUNs.

Instead of using OPENR to open a file, ENVI provides several different types of library routines for opening files. Each of these routines will return an FID for the file that was opened. The FID is then passed into the ENVI processing routine that needs access to the data. The ENVI library also provides a routine that simply reads data from the FID’s file into an IDL array that can be used when the programmer needs direct access to the data. In a similar fashion, when the results of an ENVI processing routine include an output image—whether saved to disk or memory—the FID for the result is returned by the ENVI routine.

The ENVI and IDL Library Directories

ENVI and IDL both have a directory called LIB, although the purpose of these two directories is quite different so it is important to choose the right one when saving your procedure files. The ENVI library contains the IDL code used for some of ENVI’s routines. These procedure files are provided by the ENVI developers simply as examples, however they are not actually used by the ENVI program. That is, if you edit the code in one of the files in this directory, running the corresponding program from the ENVI menu will not reflect these changes. The ENVI library directory is not in IDL’s default path. You will find the ENVI library directory directly under the main ENVI directory. For example on a typical Windows PC it will be in:

```
C:\rsi\idlXX\products\enviXX\lib
```

The IDL library directory, on the other hand, contains procedure files that are used by IDL. For example, the common IDL function called CONGRID (which resizes an array) is a built-in part of IDL, but is written in the IDL language and stored in IDL’s library directory. Thus, the IDL library directory is always in IDL’s default path. The location of the IDL library is within the main IDL directory. For example on a typical Windows PC it will be in:

```
C:\rsi\idlXX\lib
```

Of course, you can edit your IDL’s path to include a new directory, but IDL will always be able to locate any file that is saved into its LIB directory (without any special editing of the IDL path).
Common Keywords for ENVI Processing Routines

As you become more experienced using ENVI processing routines you will notice that there are several keywords that are common to virtually every routine. These keywords control the basic file input and output for the processing.

**FID**

The “File ID” is a long integer scalar. The FID is provided to the ENVI programmer as a named variable by one of several ENVI routines that can be used to open or select a file. All processing of the file using ENVI’s routines is accomplished by referring to its FID. However, if you must work with the file directly in IDL, the FID is not equivalent to a logical unit number.

**R_FID and M_FID**

ENVI processing routines that result in new images will also have an R_FID, or “Returned FID” keyword. If the results are saved to memory only, setting the R_FID keyword is the only method for accessing the data. Processing routines that allow masking will have a M_FID, or “Mask FID” keyword for specifying the file to use as the mask band.

**What if the FID is Set to -1?**

If there was a problem opening or accessing the file, the FID will be set to a value of -1. FIDs with a value of -1 are ‘invalid’ and can’t be further processed. When programming in ENVI, always check for the possibility of FID, R_FID, and M_FID being invalid. When an invalid file ID is encountered, you will usually just return from the current processing routine.

**DIMS**

The ‘Dimensions’ keyword is a long integer array with 5 elements, and defines the spatial subset (of a file or an array) to be used for processing. Almost every time an FID is specified, you will also have to specify what spatial subset of this file to use (even if the entire file, with no spatial subsetting, is to be processed).

```
DIMS[0] = a pointer to an open Region of Interest, used only in cases where ROIs define the spatial subset, otherwise set to -1
DIMS[1] = the starting sample number (an IDL zero-based array subscript)
DIMS[2] = the ending sample number
DIMS[3] = the starting line number
DIMS[4] = the ending line number
```
POS

The POS keyword defines the band positions to be used for processing, and is a long integer array of variable length. Because files that ENVI processes may have multiple bands, the spectral subset of bands to be used for the processing is specified by the POS vector. Bands are specified starting at zero (Band 1 = 0, Band 2 = 1, etc.). For example, to process only Bands 3 and 4 of a multiband file, POS = [2, 3].
Opening and Managing Files

ENVI’s routines for handling files were designed to give the programmer a great deal of flexibility. There are several routines that can be used depending on the degree of user interaction desired.

**ENVI_PICKFILE**

This routine produces a widget dialog box that prompts the user to choose a file on disk. ENVI_PICKFILE produces the same widget interface as choosing File > Open Image File from the Main ENVI menu. This routine doesn’t actually open a file, but instead returns the fully qualified pathname of the file as a string. It is often used when you know that the user will be opening a new file from disk, or when you do not intend to use ENVI routines to process the file (e.g., when you just need to get the name of the file).

**ENVI_SELECT**

This routine produces a widget dialog box that prompts the user to select a file from among those that have already been opened. ENVI_SELECT produces ENVI’s standard file selection dialog, including working buttons for spatial and spectral subsetting, as well as choosing a mask band. This routine also incorporates the functionality of ENVI_PICKFILE because the GUI interface it produces includes a button to open ENVI format files from disk. ENVI_SELECT not only returns an FID for the selected file, but also the DIMS and POS variables that will likely be required for further processing.

**ENVI_OPEN_FILE**

This routine will return an FID with user interaction and it is the most direct, simple way to open an ENVI file. The keyword NO_REALIZE will prevent the Available Bands List dialog window from being opened when the file is opened. (If the Available Bands List is already open this keyword has no effect.)

**ENVI_FILE_MNG**

This routine allows opened files to be closed and/or deleted from the disk. No user interaction is required.
ENVI_GET_FILE_IDS

This routine will return all of the currently open FIDs.

Opening External File Types

ENVI can read a wide variety of formats, although ENVI_OPEN_FILE will only open files for which there is an accompanying ENVI header file. The ENVI library contains a special processing routine to open and return an FID for external format files:

ENVI_OPEN_DATA_FILE

This routine has a keyword for each type of external file format that ENVI can read. It will return an FID and requires no user interaction.
Accessing Image Data

When image files are very large (as is often the case with remotely sensed images), it is not advisable to read in an entire file into memory all at once using IDL’s READU procedure. Instead, ENVI provides two processing routines that can be used to read image data in smaller, more manageable pieces. The two routines provide data in a logical organization—either one band at a time, or one spectral slice at a time.

**ENVI_GET_DATA**

This function is designed to retrieve spatial image data. It will return data from only one specified band at a time, so if spatial data are required from more than one band, it must be called multiple times. The dimensions returned are controlled by the DIMS keyword.

**ENVI_GET_SLICE**

This function is designed to retrieve spectral image data. It will return data from all of the image bands for one specified line, for any number of samples in that line. The data can be returned in either BIP or BIL storage order.

**Spatial Subsetting Using ROIs**

In many ENVI routines you have the option to spatially subset an image using an ROI instead of specifying a range of samples and lines. In ENVI programming routines, the DIMS variable is used to define the spatial subset. The first element of the DIMS variable is called the ROI pointer and is used to tell ENVI if the spatial subset should be based on an ROI. When the ROI pointer is set to a value of -1L, it means an ROI is not being used. Properly setting the ROI pointer is accomplished using the routine ENVI_GET_ROI_DIMS_PTR.
Image Output: Making ENVI Format Files

ENVI’s image format is perhaps the simplest format possible. It is a ‘flat binary’ file in which the raster image data are stored as a binary stream of bytes in either BSQ, BIL, or BIP storage order. This file contains only image data—header informations is not embedded in the file. IDL automatically writes this flat binary format when two- or three-dimensional arrays of image data are written to disk with the WRITEU procedure. An ENVI image data file can have any name and does not have a required filename extension.

Accompanying each flat binary file is an ASCII header file that specifies the image’s basic characteristics (such as its dimensions and data type) as well as any additional information about the file (such as its georeferencing data and sensor type). In order for ENVI to recognize the header file, it must have the same root name as the data file and a .hdr file extension. The format of the header file is detailed in Appendix B, “ENVI File Formats” of the ENVI User’s Guide.

Saving Image Data to Memory

When a user function results in image data contained in an IDL array, these data can readily be entered into ENVI as a memory-only item.

**ENVI_ENTER_DATA**

Use this routine to enter image data from an IDL array into the Available Bands List. This routine automatically sets up an ENVI header file for the image that is also stored only in memory, and will return the FID for the in-memory image. Once the image appears in the Available Bands List it can be used as any other ENVI image and can even be saved to disk using the File > Save File As > ENVI Standard routine.

Saving Image Data to Disk

Because IDL’s WRITEU procedure produces ENVI format files, ENVI does not provide a separate library routine for writing image data contained in IDL arrays to disk. Instead, just use WRITEU:

```idl
OpenW, unit, 'new_envi_image_file.img', /Get_LUN
WriteU, unit, image_array
Free_LUN, unit
```

However, it is considerably more complicated to write the ENVI header file that must accompany the data file.
ENVI_SETUP_HEAD

Use this routine to write the ENVI Header file for an image that is already saved to disk. The OPEN keyword to his routine also allows the image file to be opened into the Available Bands List. If ENVI_SETUP_HEAD is called without setting the OPEN or WRITE keywords, the ENVI header file will be created in memory only (which allows ENVI_FILE_QUERY to be used on the file’s header information throughout the rest of the user code). ENVI_SETUP_HEAD will optionally return an FID for the image file on disk.

Creating New Files from Existing ENVI Files

There is a third ENVI routine that will create new ENVI format files (including the header files) although it can only be used with files that are already opened in ENVI (i.e., files for which an FID has already been obtained).

CF_DOIT

Use this library routine to create a new ENVI format file from existing ENVI files. The images that are incorporated into the new file can be any combination of opened ENVI files on disk or in memory, and the resulting new file can also be saved to either file or memory. CF_DOIT is the equivalent routine to that in the ENVI menu under File > Save File As > ENVI Standard.
Chapter 2:  
Band and Spectral Math Functions

This chapter covers the following topics:

<table>
<thead>
<tr>
<th>Band Math Basics</th>
<th>43</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writing Band Math Functions</td>
<td>44</td>
</tr>
<tr>
<td>Compiling Band Math Functions</td>
<td>44</td>
</tr>
<tr>
<td>Example: Band Math Function</td>
<td>45</td>
</tr>
<tr>
<td>Spectral Math Basics</td>
<td>47</td>
</tr>
<tr>
<td>Writing Spectral Math Functions</td>
<td>48</td>
</tr>
<tr>
<td>Compiling Spectral Math Functions</td>
<td>48</td>
</tr>
<tr>
<td>Example: Spectral Math Function</td>
<td>48</td>
</tr>
</tbody>
</table>
Band Math™ and Spectral Math™ provide one of the simplest programming interfaces for adding processing functionality. Creating a single function for processing spatial (Band Math) or spectral (Spectral Math) data allows extensions of ENVI through these interfaces. Math functions do not require you to change menus, create processing parameter widgets, perform I/O, or other items necessary in a processing routine. Instead ENVI does all the work and you can concentrate on the processing function.
Band Math Basics

ENVI’s Band Math™ function can be used to call a custom processing function you have written. The Band Math interface is used to define the bands or files used as input, call your function, and write the result to a file or memory. Band Math accesses data spatially by mapping the variables to bands or files. Spatial data that are too large to read entirely into memory are automatically accessed using ENVI’s data tiling. See Band Math in the ENVI User’s Guide for details on using Band Math.

A Band Math function is written using variables named b1 (or B1), b2, etc. The function is called by entering its name and variables in the Band Math expression text box. The variables are assigned to bands or files using the Band Math dialog.

*Figure 2-1* depicts the band math processing for the addition of three bands. Each band in the expression is mapped to an input image band, summed together and output as the resulting image data. Any one or more of the expression’s variables can be mapped to a file instead of mapping each variable to a single band. The resulting output is a new image file. For example, in the expression b1 + b2 + b3, if b1 is mapped to a file and b2 and b3 are mapped to a single band then the resulting image file contains the bands of the b1 file summed with b2 and b3.

*Figure 2-1: The band math processes for the addition of three bands.*
With this understanding it is easy to develop functions that execute as expressions in the band math dialog. For example, to execute a function called BM_RATIO with two input bands, enter the following into the Band Math expression text box.

\[
\text{bm\_ratio}(b1, \ b2)
\]

The function declaration for this routine is

\[
\text{function bm\_ratio, } b1, \ b2
\]

The processing performed within the Band Math function has the same constraints as Band Math expressions. Input data are tiled and therefore functions like min() and max() are invalid since they would return only the minimum or maximum of the current tile and not the minimum or maximum of the input band.

**Writing Band Math Functions**

Band Math functions are simple to write and execute as Band Math expressions. The function accepts the input bands, processes the data, and returns the result. Functions follow the model:

\[
\text{function bm\_func, } b1, \ [b2, \ldots, \ bn, \ \text{parameters and keywords}]
\text{processing steps}
\text{return, result}
\text{end}
\]

**Note**

To be compatible with tiled processing, custom Band Math functions should avoid processing that requires the entire band in memory at one time.

**Compiling Band Math Functions**

Once the function is completed, the resulting .pro or .sav file should be placed in the save_add directory. This allows auto-compiling or restoring when ENVI is started. Alternately, the function can be compiled by selecting *File > Compile IDL Module* from the ENVI main menu.

**Note**

Custom Band Math functions cannot be compiled with ENVI RT. If you have ENVI RT, you must use a compiled (.sav) file to add a custom Band Math function.
Example: Band Math Function

The following example creates a custom Band Math function to perform the following ratio and optionally check for divide-by-zero.

\[(\text{b1} + \text{b2}) / (\text{b1} - \text{b2})\]

The function is declared with two input bands, \(\text{b1}\) and \(\text{b2}\), and a keyword check.

\[
\text{function bm\_ratio, b1, b2, check=check}
\]

The denominator is calculated

\[
\text{den} = \text{float(b1)} - \text{b2}
\]

and if the keyword check is set, find the location of all zeros.

\[
\text{if (keyword\_set(check)) then ptr = where(den eq 0., count) }
\]

\[
\text{else count = 0}
\]

Denominator values are temporarily set to 1.0 preventing the trap handler from being called when a divide by zero occurs.

\[
\text{if (count gt 0) then den[ptr] = 1.0}
\]

Note: The program will not crash if a divide by zero occurs but it is faster to set the values to 1.0 to avoid trap-handler overhead.

The remaining ratio is computed

\[
\text{result} = (\text{float(b1)} + \text{b2}) / \text{den}
\]

and if there were any divide by zeros the result is set to 0.0

\[
\text{if (count gt 0) then result[ptr] = 0.0}
\]

Finally, the result is returned from the function

\[
\text{return, result}
\]

The example Band Math function is shown below

\[
\text{function bm\_ratio, b1, b2, check=check}
\]

\[
\text{den} = \text{float(b1)} - \text{b2}
\]

\[
\text{if (keyword\_set(check)) then ptr = where(den eq 0., count) }
\]

\[
\text{else count = 0}
\]

\[
\text{if (count gt 0) then den[ptr] = 1.0}
\]

\[
\text{result} = (\text{float(b1)} + \text{b2}) / \text{den}
\]

\[
\text{if (count gt 0) then result[ptr] = 0.0}
\]

\[
\text{return, result}
\]

end
The following steps outline the procedure for executing this example with and without divide by zero checking:

1. Save the function as mfband.pro and place in the save_add directory.
2. Start or restart ENVI.
3. Open an input file that contains two bands.
5. To perform the ratio without divide by zero checking, enter the following expression in the “Enter Expression” text box.
   
   \[
   \text{bm\_ratio}(b1, b2)
   \]

6. To perform the ratio with divide by zero checking enter the following expression.
   
   \[
   \text{bm\_ratio}(b1, b2, /check)
   \]
Spectral Math Basics

ENVI’s Spectral Math™ function can be used to call a custom processing function you have written. The Spectral Math interface is used to define the spectra or files used as input, call your function, and write the result to a plot window, file, or memory. Spectra to be used as input must be displayed in a plot window. Spectral Math accesses data spectrally by mapping the variables to spectra or files. See Spectral Math in the ENVI User’s Guide for details on using Spectral Math.

A Spectral Math function is written using variables named s1 (or S1), s2, etc. The function is called by entering its name and variables in the Spectral Math expression text box. The variables are assigned to spectra or files using the Spectral Math dialog. Figure 2-2 depicts the Spectral Math processing for the addition of three spectra. Each spectrum in the expression is mapped to an input spectrum, summed together and the resulting spectrum is output to a plot window. Any one or more of the expression spectra may be mapped to a file instead of mapping each input to a single spectrum. The resulting output is a new image file. For example, in the expression s1 + s2 + s3, if s1 is mapped to a file and s2 and s3 are mapped to single spectra then the resulting image file contains the spectra of the s1 file summed with s2 and s3.

**Figure 2-2: The spectral math process for the addition of three spectra.**

With this understanding it is easy to develop functions that execute as expressions in the Spectral Math dialog. For example, to execute a function called SM_SCALE with two input spectra, enter the following into the Spectral Math expression text box.

```
sm_scale(s1, s2)
```

and the function declaration for this routine would be

```
function sm_scale, s1, s2
```
Note
The processing performed within the Spectral Math function has the same constraints as Spectral Math expressions. The result of the processing must be a spectrum or spectra with the same number of bands.

Writing Spectral Math Functions

Spectral Math functions are simple to write and execute as Spectral Math expressions. The function accepts the input spectra, processes the data, and returns the result. Functions follow the model:

```
function sm_func, s1, [s2,..., sn, parameters and keywords]
  processing steps
  return, result
end
```

The output of a spectral math function is a single spectrum or spectra with the same number of bands as the input. When an input parameter is mapped to a file a whole line of spectra are processed at a time.

Compiling Spectral Math Functions

Once the function is completed, it is recommended that you place the resulting .pro or .sav file in the save_add directory. This allows auto-compiling or restoring when ENVI is started. Alternately, the functions can be compiled by selecting File > Compile IDL Module from the ENVI main menu.

Note
Custom Spectral Math functions cannot be compiled with ENVI RT. If you have ENVI RT, you must use a a compiled (.sav) file to add Spectral Math functions.

Example: Spectral Math Function

The following example will create a custom Spectral Math function that performs the following ratio and optionally checks for divide-by-zero.

```
s1 / s2
```

The function is declared with two input spectra, s1 and s2, and a keyword check.

```
function sm_ratio, s1, s2, check=check
```
If the keyword check is set, find the location of any zeros.

```plaintext
if (keyword_set(check)) then ptr = where(s2 eq 0., count) $
else count = 0
```

Zero values are temporarily set to 1.0 preventing the trap handler from being called when a divide by zero occurs.

```plaintext
if (count gt 0) then s2[ptr] = 1.0
```

**Note**

The program will not crash if a divide by zero occurs but it is faster to set the values to 1.0 to avoid trap-handler overhead.

The ratio is computed:

```plaintext
result = float(s1) / s2
```

and if there were any divide by zeros the result is set to 0.0.

```plaintext
if (count gt 0) then result[ptr] = 0.0
```

Finally, the result is returned from the function:

```plaintext
return, result
```

The example Spectral Math function is shown below.

```plaintext
function sm_ratio, s1, s2, check=check
    if (keyword_set(check)) then ptr = where(s2 eq 0., count) $
    else count = 0
    if (count gt 0) then s2[ptr] = 1.0
    result = float(s1) / s2
    if (count gt 0) then result[ptr] = 0.0
    return, result
end
```

To execute the function with and without divide by zero checking:

1. Save the function as mfspec.pro and place in the `save_add` directory.
2. Start or restart ENVI.
3. Open an input file that has multiple bands.
4. Display the image, then plot two spectra by selecting *Tools > Profiles > Z Profile (Spectrum)* from the Display menu.
5. Select *Spectral Math* from the *Spectral Tools* pulldown menu.
6. To perform the ratio without divide by zero checking enter the following expression in the Enter Expression text box.

\[ \text{sm\_ratio}(s1, s2) \]

7. To perform the ratio with divide by zero checking enter the following expression

\[ \text{sm\_ratio}(s1, s2, /check) \]
Chapter 3:
Batch Processing

This chapter covers the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Mode ENVI</td>
<td>53</td>
</tr>
<tr>
<td>Initiating Batch Mode</td>
<td>55</td>
</tr>
<tr>
<td>Writing Batch Mode Programs</td>
<td>57</td>
</tr>
<tr>
<td>ENVI Message Logging in Batch Mode</td>
<td>61</td>
</tr>
<tr>
<td>Helpful Tips for Batch Processing</td>
<td>62</td>
</tr>
<tr>
<td>Examples of ENVI Batch Programming</td>
<td>65</td>
</tr>
</tbody>
</table>
About Batch Processing

Many of the ENVI image processing programs that do not require extensive user interaction were designed to be run in batch mode as well as from the menu system.

Execution in batch mode is accomplished by building a procedure containing all of the program calls and the appropriate parameters. A batch routine can be run from the menu system with minimal GUI input or external to the menu system with no GUI input. The first way allows a combination of processes to be linked together and started from a single menu selection. The second technique allows for common processing steps to be performed on any number of files external to the ENVI graphical interface.

**Note**

The execution of procedures from the IDL command line is not available if you have purchased ENVI RT (an ENVI installation that uses the run-time version of IDL). ENVI RT can only be extended by the inclusion of compiled (.sav) files.
Batch Mode ENVI

Running ENVI in batch mode allows you to use ENVI in a command line mode. This capability can be useful in several cases: if you are primarily working in IDL but occasionally need to use ENVI routines; if you want to create custom procedures that combine your own IDL code with ENVI routines; or, if you need to do a large amount of ENVI processing without any user interaction (i.e., go home and have the ENVI processing done overnight).

Running ENVI in batch mode is no different than working in an ordinary IDL session, except that ENVI’s processing routines are accessible as a special library of IDL procedures and functions. In order to gain access to these library routines they must be restored into the IDL session’s RAM. However, proper access to all of the ENVI library routines from IDL requires some knowledge of how the ENVI program is structured.

ENVI is broken into approximately 40 small IDL “save” files, binary format files that contain data and compiled routines. These save files are stored in a directory called SAVE that is within the main ENVI directory. The core ENVI save files contain the basic ENVI library functions, or programming routines, and the internal variables required to run ENVI. On a typical Windows PC you would find the files in c:\rsi\idlxx\products\envi\save. For a typical UNIX installation, they would be located in /usr/local/rsi/idl_x.x/products/envi_x.x/save. When a normal ENVI session is started, only a small subset of these save files, which enable the core ENVI functionality, are restored.

Running ENVI in batch mode requires restoring these core save files. Then, a special ENVI routine named ENVI_BATC_INIT is called, which “turns on” the batch mode session. The combined process is called initiating Batch Mode.

Hybrid Batch Mode

Remember, ENVI is just an IDL program. If you use the IDL session that’s also running ENVI then you already have access to all of the ENVI procedures and functions that have been restored. This state is often referred to as hybrid batch mode because you can use ENVI-specific library routines without having to initiate batch mode. This can be both convenient and problematic! For example, if an IDL procedure you’re running at the command line produces a new image band, you could directly enter these new data into the Available Bands List for use in the ENVI session by using the library routine ENVI_ENTER_DATA. However, if the IDL procedure crashes, then you’ve also crashed the concurrent ENVI session! When developing ENVI user functions it is often convenient to work in hybrid batch mode.
because it simulates the environment where the code will eventually be executed; however, when running true batch processes we recommend that you use a separate IDL session where batch mode has been initiated.

In order to use hybrid batch mode you must have a full version of ENVI that includes IDL (i.e., you can’t be in ENVI Runtime) and the IDL command line must be active. The control for blocking the IDL command line is in the ENVI Configuration File and can be changed from within ENVI by selecting File > Preferences, clicking the “Misc. System Preferences” button, and clicking the “Command Line Blocking” arrow toggle button to display “NO.”
Initiating Batch Mode

Initiating ENVI batch mode requires restoring several of ENVI’s binary save files and then calling an ENVI command called ENVI_BATCH_INIT.

1. Start ENVI.
2. From the ENVI main menu, select File > Preferences.
3. Click “Misc. System Preferences” and ensure that the “Exit IDL on Exit from ENVI” arrow toggle button is set to “No.”
4. Exit ENVI and exit IDL.

**Warning**

Do not attempt to initialize ENVI in batch mode from an IDL session that is currently running an interactive ENVI session. Instead, start a new IDL session to initialize ENVI in batch mode.

5. Start a new IDL session and enter the following command:

   IDL> ENVI, /restore_base_save_files

   If you forget to use the keyword RESTORE_BASE_SAVE_FILES, you will end up starting a normal ENVI session.

   If you have multiple versions of IDL installed on your computer you will have to be careful that the IDL executable you are using to start a new IDL session was the one associated with your ENVI software when installed. That is, the IDL executable that is being used must be the one in which the main ENVI directory is installed. For example, in a standard UNIX installation, the ENVI executable file is:

   `/usr/local/rsi/idl_x.x/products/envi_x.x/bin/envi.run`

   The path to the IDL executable in the same installation would be:

   `/usr/local/rsi/idl_x.x/bin`

   This IDL installation includes an extra file (the envi.sav file) in its idl_x.x/lib/hook directory. If you try initiating batch mode from a separate installation of IDL (i.e., if you’re not using the IDL in which ENVI was installed), you will receive the error “Attempt to call undefined procedure/function: ENVI.”

6. Next, call the ENVI__BATCH_INIT procedure:

   IDL> ENVI_BATCH_INIT
Chapter 3: Batch Processing

Calling ENVI_BATCH_INIT is nearly identical to starting a new ENVI session except that there’s no GUI. After batch mode has been established, all subsequent calls to ENVI_BATCH_INIT are ignored.

The following is an example code for initiating batch mode.

```
; *******************************************************
; This batch example shows how to initialize ENVI
; in batch mode.
;
; For more information see the ENVI Programmer's Guide.
; *******************************************************
; Copyright (c) 2000-2001, Research Systems Inc.
; *******************************************************
pro bt_init
  envi, /restore_base_save_files
  envi_batch_init, log_file='batch.log'

; Batch processing would go here

  envi_batch_exit
end

Exiting Batch Mode

If you are going to continue working in the IDL session after you’ve finished ENVI batch mode work, then it’s equally important to exit the batch mode session properly. As you might imagine, in order to run a program as complex as ENVI, many different variables, common blocks, structures, pointers, and objects are created. When you exit ENVI, all of these components are properly deleted and the memory they consume is released. This is quite important if you’re going to continue working in the same IDL session. The command ENVI_BATCH_EXIT has the same effect in batch mode as choosing File > Exit during an ordinary ENVI session. For example, the license used for the ENVI session will be released.
Writing Batch Mode Programs

Using ENVI Routines in IDL Programs

The primary purpose of ENVI’s batch mode is to allow ENVI processing without user interaction. Many users also find it convenient to add functionality to their own stand-alone IDL programs by using some ENVI library routines. If ENVI has a function you’d like to use, why not just use it instead of coding it from scratch? Of course, in order to gain access to the library routines, the IDL session where the program is running must be in batch mode. It is recommended that the commands for initiating batch mode be included within the IDL program to ensure that batch is initiated and that ENVI_BATC_H_EXIT is called upon completion to clean up ENVI-specific resources that may consume memory.

Example: Writing a Simple ENVI Batch Mode Procedure:
VIEW_DEM

This example shows a simple IDL procedure that prompts the user to select a DEM file, then displays the DEM and its shaded relief image side-by-side. Instead of coding a shaded relief algorithm from scratch, we use ENVI’s library routine ENVI_DOIT with the TOPO_DOIT keyword to compute the shaded relief image.

Note

If a previous IDL session is open, exit and start a new IDL session before running this example code.

```idl
PRO VIEW_DEM

dem_file = ENVI_PICKFILE(TITLE = 'select a DEM')
IF (dem_file EQ "") THEN RETURN
ENVI_OPEN_FILE, dem_file, R_FID = dem_fid

ENVI_FILE_QUERY, dem_fid, NS = ns, NL = nl
proj = ENVI_GET_PROJECTION(FID = dem_fid, PIXEL_SIZE = pixel_size)
dims = [ -1L, 0, ns - 1, 0, nl - 1 ]
ENVI_DOIT, 'TOPO_DOIT', AZIMUTH = 15.0, BPTR = [2], DIMS = dims, $ ELEVATION = 45.0, FID = dem_fid, IN_MEMORY = 1, POS = [0], $ R_FID = shaded_fid, PIXEL_SIZE = pixel_size

dem = ENVI_GET_DATA(FID = dem_fid, DIMS = dims, POS = [0])
shaded = ENVI_GET_DATA(FID = shaded_fid, DIMS = dims, POS = [0])
```
Writing Batch Mode Programs

Note
See the following section to understand why you got syntax errors when the code was compiled.

Example: Using COMPILER_OPT

If you save and compile the code used in the previous example, you will receive a number of IDL compilation errors. The lines where errors occurred all contain ENVI library functions. Because these functions are not a built-in part of IDL, the IDL compiler doesn’t recognize them as functions and instead assumes that they’re variables that are being dereferenced. This problem arises because IDL originally allowed function calls and variable dereferencing to use the same syntax:

```idl
number = my_array(0,0)
result = my_function(0,0)
```

In modern versions of IDL (release 5.0 and later) the syntax for dereferencing variables changed to use square brackets instead of parentheses. This newer syntax eliminates the ambiguity, but to ensure backwards compatibility of IDL code the compiler still must recognize both types of syntax. Previously the only solution for this problem was using the FORWARD_FUNCTION statement to declare the names of uncompiled functions so that the compiler would correctly recognize them. In IDL 5.3, a much better solution was introduced. Using the COMPILER_OPT statement, you can instruct the IDL compiler to strictly enforce the new square brackets syntax for dereferencing variables, thus allowing the compiler to identify previously unknown functions correctly.

1. Edit the `view_dem.pro` file to include the following line (shown in bold in the following code) by inserting it immediately after the procedure definition statement.

```idl
pro VIEW_DEM
   compile_opt STRICTARR
```

The procedure will now compile and run.
Using ENVI Recording to Write Batch Code

The ENVI Log Manager can be used in interactive ENVI to save an ASCII file containing information about each processing function called and its parameters (for more information, see the online help or the *ENVI User’s Guide*. (Vector and matrix parameters are currently not logged.) The recording file can aid the user in creating batch routines. Portions of the ASCII file can be used as step-by-step instructions for a batch routine to perform the same processing.

The following logfile example lists information about the following processes completed in one ENVI session:

- The opening of a file
- ISODATA classification
- Class sieve
- Class clump

Each step in the log file is separated by three asterisks and a blank line.

```plaintext
*** Opened File: E:\DATA\canyon.tm [Thu Nov 13 09:19:49 1997]
***

*** Classification *** [Thu Nov 13 09:22:06 1997]
Method: IsoData
Input File: E:\DATA\canyon.tm
Bands: 1-6
Dims: 1-640,1-400
Output File: To Memory
Output Rule File: NONE
Number of Classes: 7
Change Threshold: 5.00
Iterations: 1
***

*** Sieve Classes *** [Thu Nov 13 09:25:35 1997]
Input File: (M8) (640x400x1)
Bands: 1
Dims: 1-640,1-400
Selected Classes: 1-7
Group Minimum Threshold: 15
Number of Neighbors : 8
Output File: To Memory
***

*** Clump Classes *** [Thu Nov 13 09:26:07 1997]
Input File: (M9) (640x400x1)
```
For this example, each step after the file open could be accomplished in batch mode using a call to an ENVI_DOIT routine. The code for the equivalent batch routine would use the following processing functions:

- Initialize ENVI in batch mode, `ENVI_BATCH_INIT`
- Open the input file, `ENVI_FILE_OPEN`
- Perform the ISODATA classification, `ENVI_DOIT` using the `CLASS_DOIT` keyword
- Sieve the classification image, `ENVI_DOIT` using the `CLASS_CS_DOIT` keyword
- Clump the classification image, `ENVI_DOIT` using the `CLASS_CS_DOIT` keyword
- Exit ENVI, `ENVI_BATCH_EXIT`
ENVI Message Logging in Batch Mode

Perhaps the most powerful (and most common) use for ENVI batch mode is to process data files with absolutely no user interaction. For example, if you had hundreds of image files that needed to be processed identically using ENVI, you could write an IDL procedure that finds all the files, opens them, performs all the ENVI processing, and saves the results to disk. All of this processing could occur while no one is present; all you’d have to do is call your ENVI batch mode program in IDL and go home!

Using the Batch Mode Log File

When using ENVI in batch mode to do extensive processing without any user interaction, it is important to ensure that (1) the batch processing won’t be suspended when an error or informational message is issued, and (2) important messages from the system are collected so that you can review them later, after the processing has finished. This is accomplished by initiating batch mode in a slightly different manner than we have done thus far.

Initiating Batch Mode with a Log File

When initiating batch mode you have the option of defining a log file with the LOG_FILE keyword. This keyword is used to pass ENVI a filename for a writable file that will receive any error or informational messages.

```
IDL> ENVI, /RESTORE_BASE_SAVE_FILES
IDL> ENVI_BATCH_INIT, LOG_FILE = 'test_batch_log.txt'
```

When running a real ENVI batch mode processing procedure, the batch log will accumulate any system-generated messages. You can also write your own messages into the log file using ENVI_ERROR.
Helpful Tips for Batch Processing

While each user is likely to have unique needs for batch processing, the following suggestions are often helpful:

- Always run your batch processing code in IDL, not in hybrid batch mode. When an interactive ENVI session is running from the same IDL session that is executing a batch processing program, some library routines may halt and wait for user input.

- It can be quite helpful to follow a standard convention for file naming, especially if the data to be processed consist of multiple files (image data, leader, trailer, navigation data, engineering data, etc.) but only certain types will be processed by your batch code.

- Use the IDL function FINDFILE to make a list of the files to process. It is recommended that full path names be used.

- Using IDL’s string operators such as STRMID, RSTRPOS, and STRSPLIT you can usually construct intuitive output filenames from the names of the input files. For example, if the input file pathname (in the variable IN_FNAME) is `c:\data\my_image_L0.dat`, and you want to name the output file (OUT_FNAME) `c:\data\my_image_L1.img` you could use the following command:

  ```idl
  out_fname = $ STRMID(in_fname, 0,STRPOS(in_fname,"_",/reverse_search))+'_L1.img'
  ```

Making a Shortcut for Initiating Batch Mode

If you frequently work in Batch Mode, you may find it convenient to put the commands that initiate Batch Mode into an IDL file so that you can initiate Batch Mode with a single command. Here is a simple example:

```idl
PRO ENVI_BATCH
   ENVI, /RESTORE_BASE_SAVE_FILES
   ENVI_BATCH_INIT
END
```

You could then initiate batch mode by simply calling your shortcut at the IDL command line:

```idl
IDL> envi_batch
```

If you would like to make a shortcut that is a bit more flexible and offers an option to initiate batch mode with a batch log file, you can do this with a slightly more
sophisticated shortcut procedure. An example of one such routine called START_BATCH is included in the example code below. To initiate batch mode with a log file, simply call START_BATCH with the name of the batch file as its first argument and a variable that will receive the batch log’s LUN as its second argument:

```idl```
IDL> START_BATCH, "my_batch_log_filename.txt", batch_unit
```idl```
To initiate batch mode without a batch log file, call START_BATCH with no arguments:

```idl```
IDL> START_BATCH
```idl```
The following is the example code for the START_BATCH.pro procedure.

```idl```
;------------------------------------------------------+
PRO START_BATCH, batch_log_name
;
; An example shortcut procedure for initiating ENVI batch mode.
;
IF (N_PARAMS() GT 1) THEN BEGIN
  PRINT, 'ERROR: Too many parameters.'
  PRINT, 'START_BATCH cannot be called with more than' + $
  ' 1 parameter.'
  RETURN
ENDIF

; if a batch log file is requested then make sure the
; filename is valid (i.e., a scalar string) and then
; initiate batch mode
IF (N_PARAMS() EQ 1) THEN BEGIN
  sz = SIZE(batch_log_name)
  IF (sz(0) NE 0) OR (sz(1) NE 7) THEN BEGIN
    PRINT, 'ERROR: Filename argument must be a scalar string.'
    RETURN
  ENDIF
  ENVI, /RESTORE_BASE_SAVE_FILES
  ENVI_BATCH_INIT, LOG_FILE = batch_log_name
ENDIF ELSE BEGIN
; if no batch file is requested then just initiate batch mode
ENVI, /RESTORE_BASE_SAVE_FILES
ENVI_BATCH_INIT
ENDELSE
END
Examples of ENVI Batch Programming

The following examples illustrate ENVI batch routines. The first example computes statistics on an input file and does not link together multiple ENVI processing routines. The next example, links together multiple processing steps and uses the output from the previous step as input into the next step. Examples are also included in Chapter 9, “ENVI Routines” for each processing function.

Example: File Statistics (Non-interactive)

The following example uses the non-interactive batch mode to compute the basic statistics of the specified file. First, the ENVI core files are restored and ENVI is started in batch mode, see “ENVI_BATC H_INIT” on page 341. Next the file is opened using ENVI_OPEN_FILE (see “ENVI_OPEN_FILE” on page 476) and the returned FID is passed into the statistics processing routine, ENVI_STATS_DOIT (see “ENVI_STATS_DOIT” on page 519). When the processing is completed, the ENVI session is terminated using ENVI_BATCH_EXIT (see “ENVI_BATC H_EXIT” on page 339).

The sample code listing follows and can be found in the file btstats1.pro in the lib directory of the ENVI installation.

```plaintext
; ******************************************************;
; This batch example shows how to calculate statistics;
; in ENVI batch mode.
;
; For more information see the ENVI Programmers Guide.
; ************************************************************************
; Copyright (c) 2000-2001, Research Systems Inc.
; ************************************************************************

pro btstats1

; Restore the core file and start ENVI in batch
ENVI, /RESTORE_BASE_SAVE_FILES
ENVI_BATCH_INIT, LOG_FILE = 'batch.log'

; Open the input file
ENVI_OPEN_FILE, 'e:\data\test.img', R_FID = fid
IF (fid EQ -1) THEN BEGIN
  ENVI_BATCH_EXIT
  RETURN
ENDIF

ENVI_FILE_QUERY, fid, NS = ns, NL = nl, NB = nb
```
; Set the DIMS and POS to process the entire image, all bands
dims = [-1, 0, ns - 1, 0, nl - 1]
pos = LINDGEN(nb)

; Calculate the basic statistics
ENVI_DOIT, 'environ_stats_doit', $
  FID=fid, POS=pos, DIMS=dims, $
  DMIN = dmin, DMAX = dmax, MEAN = mean, $
  STDV = stdv, COMP_FLAG = 1

; make sure each one is defined on the return
PRINT, dmin, dmax, mean, stdv

; Exit ENVI
ENVI_BATCH_EXIT
END

Additional statistics are available using ENVI_STATS_DOIT, see page 519.

The following steps outline the execution of this example:

1. Edit the paths and filenames to match those on the current machine; save the file as an IDL (.pro) file.
2. Start IDL (without ENVI running)
3. Type the following on the IDL command line (the line that contains the prompt "IDL") to compile the routine. The path to the saved routine is c:\my_path.

   .compile c:\my_path\bt_stat

4. Type the following on the ENVI command line to execute the routine.

   btstat1

5. The resulting statistics are printed to the IDL log window.

**Example: Saturation Stretch (Non-interactive)**

The following example performs a saturation stretch on an RGB image. This example is intended to be run from the command line without any user interaction. The batch routine links together a number of ENVI processing functions. The operations and corresponding functions are listed in the following steps:

1. Open an RGB image file, ENVI_OPEN_FILE
2. Perform a 2% stretch on the RGB image, STRETCH_DOIT
3. Transform the stretched RGB to HSV color space, RGB_TRANS_DOIT
4. Gaussian stretch the saturation band, STRETCH_DOIT
5. Transform the HV and stretched saturation band to RGB, RGB_ITRANS_DOIT

Non-interactive batch routines must first restore the ENVI core files. The input, output, and temporary filenames used by this example must be changed to reflect the configuration of the current machine. The input file must reference an RGB image that already exists.

ENVI is initialized in batch mode using ENVI_BATCH_INIT (see “ENVI_BATCH_INIT” on page 341). The routines, STRETCH_DOIT (see “STRETCH_DOIT” on page 687), RGB_TRANS_DOIT (see “RGB_TRANS_DOIT” on page 632), and RGB_ITRANS_DOIT (see “RGB_ITRANS_DOIT” on page 629), are restored automatically when called using ENVI_DOIT.

The processing steps in this example use the output from one “doit” as input into the next. Each processing routine creates and opens the output files allowing the return of the file id using the keyword R_FID. The file id or an array of file ids are passed into the next processing routine.

The sample code listing follows and can be found in the file btsat.pro in the lib directory of the ENVI installation.

```idl
; *******************************************************
; This batch example is run from the IDL command line without any
; user interactions.
;
; This batch routine performs a saturation stretch on an rgb file.
;
; 1. Open an rgb file.
; 2. Perform a 2% stretch on the rgb, store the result in tmp1_name
; 3. Transform the rgb to hls, store the result in tmp2_name.
; 4. Gaussian stretch the saturation band, store the result
; tmp3_name.
; 5. Transform the hl and stretched saturation band to rgb, store
the
; result in out_name.
;
; For more information see the ENVI Programmers Guide.
; *******************************************************

; Copyright (c) 2000-2001, Research Systems Inc.
; *******************************************************
PRO satstrch

; Restore the ENVI core files
```
ENVI, /RESTORE_BASE_SAVE_FILES

; Initialize ENVI and send all errors to an error file.
ENVI_BATCH_INIT, LOG_FILE = 'e:\data\testing\batch.log'

; Define the needed file names (remember to specify the full path).
in_name = 'e:\data\testing\test.img'
out_name = 'e:\data\testing\new_rgb'
tmpl_name = 'e:\data\testing\tmpl1'
tmp2_name = 'e:\data\testing\tmpl2'
tmp3_name = 'e:\data\testing\tmpl3'

; open the file
ENVI_OPEN_FILE, in_name, R_FID = fid
IF (fid EQ -1) THEN BEGIN
   ENVI_BATCH_EXIT
   RETURN
ENDIF

; Set up to process the entire image, first 3 bands as RGB
ENVI_FILE_QUERY, fid, NS = ns, NL = nl, BNAMES = bnames
dims = [-1, 0, ns - 1, 0, nl - 1]
pos = [0, 1, 2]

; Stretch the input image with a 2% stretch
ENVI_DOIT, 'stretch_doit', FID = fid, POS = pos, DIMS = dims,
   OUT_NAME = tmpl_name, METHOD = 1, OUT_DT = 1,
   I_MIN = 2.0, I_MAX = 98.0, RANGE_BY = 0,
   OUT_MIN = 0, OUT_MAX = 255, IN_MEMORY = 0,
   R_FID = st_fid

IF (N_ELEMENTS(st_fid) EQ 0) THEN BEGIN
   ENVI_BATCH_EXIT
   RETURN
ENDIF

; Convert stretched data to hls, all bands are from the
; same file so make an array of 3 from st_fid
ENVI_DOIT, 'rgb_trans_doit', FID = [st_fid, st_fid, st_fid],
   POS = pos, OUT_NAME = tmp2_name, DIMS = dims, R_FID = hls_fid,
   HSV = 0, IN_MEMORY = 0

IF (N_ELEMENTS(hls_fid) EQ 0) THEN BEGIN
   ENVI_BATCH_EXIT
   RETURN
ENDIF
Chapter 3: Batch Processing

ENVI Programmer’s Guide

Examples of ENVI Batch Programming

; Gaussian stretch the saturation band, do a percent
; stretch 0% to 100% (the entire range). Set the output range
; from 0.0 to 1.0. Store the result tmp2_name.

ENVI_DOIT, 'stretch_doit', $
   FID = hls_fid, POS = [2], DIMS = dims, $
   METHOD = 3, RANGE_BY = 0, I_MIN = 0.0, $
   I_MAX = 100.0, STDV = 2.0, OUT_DT = 4, $
   OUT_MIN = 0.0, OUT_MAX = 1.0, IN_MEMORY = 0, $
   R_FID = gst_fid, OUT_NAME = tmp3_name

IF (N_ELEMENTS(gst_fid) EQ 0) THEN BEGIN
   ENVI_BATCH_EXIT
   RETURN
ENDIF

; Preform the inverse color transformation of the hl and the
; stretched saturation band back to rgb. Now we incorporate
; the results of two file for the inverse transformation and
; must build the fid and pos arrays.

out_bname = 'satstrch(' + bnames[pos] + ')

ENVI_DOIT, 'rgb_itrans_doit', $
   FID = [hls_fid, hls_fid, gst_fid], POS = [0, 1, 0], $
   OUT_NAME = out_name, DIMS = dims, HSV = 0, $
   OUT_BNAME = out_bname, IN_MEMORY = 0

; Close the input file and delete the tmp files from disk.
ENVI_FILE_MNG, ID = fid, /REMOVE
ENVI_FILE_MNG, ID = st_fid, /REMOVE, /DELETE
ENVI_FILE_MNG, ID = hls_fid, /REMOVE, /DELETE
ENVI_FILE_MNG, ID = gst_fid, /REMOVE, /DELETE

; Remember to exit envi
ENVI_BATCH_EXIT
END

The following steps outline the execution of this example:

1. Edit the paths and filenames to match those on the current machine. Save the file as an IDL (.pro) file.
2. Start IDL (without ENVI running)
3. Type the following on the IDL command line (the line that the contains the prompt “IDL>”) to compile the routine. The path to the saved routine is c:\my_path.
   .compile c:\my_path\btsat.pro
4. Type the following on the ENVI command line to execute the routine.

```
satstrch
```

5. Start ENVI and view the RGB file created (see the variable OUT_NAME for the RGB filename).
### Chapter 4: User Functions

This chapter covers the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Functions</td>
<td>73</td>
</tr>
<tr>
<td>Modifying the ENVI Menus</td>
<td>75</td>
</tr>
<tr>
<td>Adding Widgets to User Functions</td>
<td>82</td>
</tr>
<tr>
<td>An Illustrated Glossary of ENVI Widgets</td>
<td>83</td>
</tr>
<tr>
<td>Auto-Managed Widget Events</td>
<td>92</td>
</tr>
<tr>
<td>Trapping Errors in User Functions</td>
<td>98</td>
</tr>
<tr>
<td>Working With Image Displays</td>
<td>101</td>
</tr>
<tr>
<td>Using Processing Routines and Tiling</td>
<td>102</td>
</tr>
<tr>
<td>Adapting User Functions for ENVI RT</td>
<td>120</td>
</tr>
</tbody>
</table>
This chapter covers the development of custom user functions. User functions are simply IDL code in which you can call ENVI processing routines, and which are accessed through the interactive ENVI menus. When designing user functions you can choose to use no interface, use ENVI’s compound widgets to simplify interface design and give your functions the same look-and-feel as ENVI, or create your own interface using IDL widgets. If you choose to use ENVI’s compound widgets, you can let ENVI automatically manage input from your interface to your user function.
Chapter 4: User Functions

User Functions

User functions allow you to directly add new functionality to ENVI by adding your own IDL procedures to the ENVI menu. Any number of user functions can be added, with each getting its own menu item. User functions are semi-permanent (i.e., they will remain there until you choose to remove them) and are executed when the user chooses them from the menu—just like any other ENVI routine. The user function procedures are actually nothing more than IDL programs in which you can call ENVI routines. In this sense, they are nearly identical to batch mode procedures, except that there is no need to initiate batch mode (since ENVI is already running). However, an additional step is required in order to add the new routine to the ENVI menu system.

Note
Because ENVI Runtime is an IDL Runtime application, it can't compile any ASCII .pro files. For this reason, all user functions added to ENVI RT must be stored in binary .sav files. For more information on how to make save files for ENVI RT please see “Adapting User Functions for ENVI RT” on page 120.

User Functions are Widget Event Handlers

You do not need to be versed in IDL widget programming in order to add user functions to ENVI. However, because ENVI is itself an IDL widget program, it will be important to know a few widget basics.

Because user functions are additions to an IDL widget program, all user function procedures are technically event handlers, a special class of IDL procedures that are executed in response to a widget event occurring. In the case of a user function, the widget event occurs when the user chooses the user function’s menu item from the ENVI menu. While ENVI (or any widget program) is running, a special IDL routine called XMANAGER runs in the background and monitors for widget events (this is what allows widget programs to be interactive). When an event occurs, information about the event is passed into the event handler procedure by XMANAGER in the form of a structure variable called the event structure. Thus, all ENVI user function procedures must follow the one simple rule for event handlers:

The procedure definition statement for a user function must include a positional parameter to receive the event structure variable:

```idl
pro MY_USER_FUNCTION, event
```

Even though most user functions will never need to use the information contained in the event structure, the positional parameter must still be included.
The programmer defines all aspects of the routine, including the level of user interaction (from none to extensive). The user function procedure files can be either `.pro` or `.sav` files, and like any IDL program, can include calls to external C or Fortran code. Further, if the user function procedure files are placed in the `SAVE_ADD` directory, they are automatically compiled (or restored) into the ENVI session’s RAM when ENVI starts (so they are always available to any ENVI user). Also, once the user function has been added to ENVI, its code can be modified at any time, recompiled from within the current ENVI session, and then used in its modified form without having to restart ENVI.
Modifying the ENVI Menus

The ENVI menu system, comprised of the main menu that appears when you start ENVI and the Display menu, which is accessed only from image display windows, is defined by two ASCII files located in the MENU directory:

Windows:

X:\rsi\idlxx\products\envixx\menu

UNIX:

/usr/local/rsi/idl_x.x/products/envi_x.x/menu

The envi.men file defines the Main menu and the display.men file defines the Display menu. Each time a new ENVI session is started, the two menu files are read and the ENVI menus are then constructed based on the content of the files. To add a new item to the menu, simply add a new line to one of these two files and restart ENVI.

Each ENVI menu item is defined by a one line entry in one of the files. The item’s definition statement will include a number that defines the level of the menu item (i.e., how deeply it is nested within other pulldown menu items), the text that will appear on the item, a widget user value for the item, and the name of the procedure that is executed when the item is selected (i.e., the name of the user function procedure in the SAVE_ADD directory). For example, Figure 4-1 illustrates an ENVI menu item that is nested four levels deep into the File menu. An excerpt from the envi.men file that defines the highlighted items is also illustrated.

Because both menu files are editable, the entire ENVI menu system can be changed by the user. Menu items can be renamed, moved, duplicated, or completely removed altogether. Similarly, the user function’s item can be added in any location. For example, if the user function is a new filtering routine, you may choose to add it to the Filter menu.

Working with the Menu Files

Using any text editor, you can open the envi.men file located in ENVI’s MENU directory.

The top portion of the file, where each line is preceded by a semi-colon, contains a brief description of the file. Following these comment lines, each item in the ENVI main menu is defined by its own line in this file.

Refer to the line towards the top of the file where the menu item definitions begin. You should find the following line:
1 {Open Image File} {open envi file} {envi_menu_event}

Figure 4-1: An ENVI menu and a menu file example.

The number at the beginning of the line defines the hierarchy of the menu item (with zero being a main item that opens a pulldown menu, a “1” being the first level of items beneath the main, a “2” being the next, nested, menu item beneath a level 1 item, etc.).

{Open External File} The first set of curly brackets defines the text that will appear as the menu item.

{open envi file} The second set of curly brackets defines the user value assigned to the menu item. This can be used to programmatically determine what type of event occurred. The user value is typically used only when the same user function procedure handles events from more than one menu item, in which case the user value will identify which item was selected.
The third set of curly brackets defines the event handler procedure (i.e., the name of the user function procedure) to execute when the menu item is selected. You will notice that none of the event handler names include the .pro or .sav extensions. The name of the user function procedure itself should be listed here, not the name of the file that contains the procedure.

Read a little further down into the envi.men file and compare the menu definitions for the File item with the menu items you see on the Main menu under File.

Menus that contain other menus (referred to as nested menus) are created by defining only a hierarchy number and the first set of curly brackets; a separator line is created by adding the optional fourth set of curly brackets containing the word “separator.”

A Note About the Menu Item’s User Value

You may have noticed that most of the event handlers (the last set of curly brackets) for the items under the File menu are the same (ENVI_MENU_EVENT); however, the user values (the second set of curly brackets) all have unique names. For most of ENVI’s built-in functions, a single event handler procedure is used to manage all of the possible events. When an item on the File menu is selected, the event handler procedure first checks the event structure to identify the item’s user value, then skips to the appropriate section in the event handler procedure. This is a nice, clean method to use for the built-in ENVI functions because there are literally hundreds of menu items. When adding only a few user functions to ENVI, most programmers find it convenient to make a separate user function procedure for each new menu item. In this case, the contents of the second set of curly brackets are meaningless, since they are never actually referenced in the user function procedure. However, because each new menu item definition must still have this second set of curly brackets, it is often a good idea to give the user value a name that you will recognize, such as the name of the user function itself. Or, many users choose to put the text ‘not used’ or ‘dummy’ into the user value definition so that when they read through the menu file, it is immediately clear that the user value is not being used.

Example: Making a Simple User Function

The following example will help you become comfortable with modifying ENVI menus. In the example shown, we create a new menu item and a simple user function. The user function will be called when the new menu item is selected in ENVI.

Note

Because ENVI Runtime is an IDL Runtime application, it can’t compile any ASCII .pro files. For this reason, all user functions added to ENVI RT must be stored in
binary .sav files. For more information on how to make save files for ENVI RT please see “Adapting User Functions for ENVI RT” on page 120.

1. In your envi.men file, scroll to the bottom of the file. Just above the line that defines the Help menu item, make a new main level menu item to hold all of the user functions you will create:

0 {MyFunctions}

2. Nested within the MyFunctions menu item, add an item called Basic File Info.

1 {Basic File Info} {not used} {file_info}

This section of the menu file should now look like this:

0 {MyFunctions}
1 {Basic File Info} {not used} {file_info}

3. Save the modified menu file.

4. If an ENVI session is open, then close it and restart ENVI. Check to be sure that your new items are now on the menu.

5. Using the IDLDE that’s running the ENVI session you just started, open a new Editor window.

This user function will simply prompt the user to select an opened file and then print some basic information about the file to the IDL Command Log window. We will use the ENVI_FILE_QUERY routine to get image size information and the IDL routine FSTAT to get file size information. The FSTAT routine takes as its argument the IDL logical unit number for the file. Because the FID is not an IDL logical unit number, we will also have to open the file using IDL File I/O procedures.

6. Enter these lines into the new editor window:

```idl
pro FILE_INFO, event
ENVI_SELECT, title='choose a file', fid=in_fid
ENVI_FILE_QUERY, in_fid, ns=ns, nl=nl, nb=nb, fname=fname
OpenR, unit, fname, /Get_LUN
info = FSTAT(unit)
Free_LUN, unit
print, 'you selected ',fname
print, 'number of samples = ',ns
print, 'number of lines = ',nl
print, 'number of bands = ',nb
print, 'file size in bytes = ',info.size
end```

7. Save the file into the `SAVE_ADD` directory as `file_info.pro`. The default `SAVE_ADD` directory is always located under the main ENVI directory.

8. Compile the user function procedure to be sure it contains no syntax errors.

**Note**
Because you are working in hybrid batch mode, any routine you compile in IDL will also be compiled and available for use in the concurrent ENVI session.

9. In ENVI, use your new menu items to run the Basic File Info user function.

10. In the IDL Command Log window, you should see the results of our simple user function as shown here. Choose any image file that is in an ENVI-supported format.

    you selected D:\enviprog\AVIRIS\Cup95_at.int
    number of samples = 400
    number of lines = 350
    number of bands = 50
    file size in bytes = 14000000

11. Try running it again, but this time when you are prompted to select a file, click on the “Cancel” button in the file selection dialog. What happens?

    % OPENR: Filename argument must be a scalar string FNAME.
    % Execution halted at: FILE_INFO
    D:\RSI\ENVI34\save_add\file_info.pro
    % WIDGET_PROCESS_EVENTS
    % $MAIN$

    The user function crashed because we forgot to account for the possibility of the user cancelling the function!

**Recovering from a User Function Crash**

At this point, our ENVI session is inactive. It has essentially crashed. If we had already done a lot of processing, in memory, we could potentially lose all of our unsaved work. If we look at the error messages in the ENVI Command Log window, we can see that three programs are listed: `FILE_INFO`, `WIDGET_PROCESS_EVENTS`, and `$MAIN$`. This list is called the *stack trace* and tells us exactly where IDL stopped. When more than one program is listed, these other programs must have been called from within ENVI, in chronological order from the bottom up. The program called `$MAIN$` refers to the main level program, which in this case is the ENVI widget program (this is where IDL started). The program `WIDGET_PROCESS_EVENTS` is an internal ENVI procedure that is involved with executing user functions. The top procedure name in the stack trace is our user function, `FILE_INFO`. 
When a crash occurs, the ENVI cursor will be located within the program level that is listed at the top of the stack trace, at the reported line number.

1. At the ENVI command line, type:

   ```idl
   ENVI> help, in_fid, fname
   IN_FID INT = -1
   FNAME UNDEFINED = <Undefined>
   ```

   If we go back and look at our user function code we can follow the chain of events that caused the crash. Because we cancelled the ENVI_SELECT window, the returned FID was not defined (-1), so the ENVI_FILE_QUERY function failed, causing the FNAME variable to be undefined and the OPENR function to crash.

   In many cases it is possible to recover from a crash in an ENVI user function by entering RETALL at the ENVI command line. This command is short for “return all” and instructs IDL to return to the $MAIN$ program level (where ENVI is active).

2. In the ENVI Command Log window, type:

   ```idl
   ENVI> retall
   ```

Checking Errors

Now that your user function has been added into the ENVI menu file, we can modify the code, recompile it, and use the new version in the current ENVI session without having to restart.

1. Return to the IDE Editor window where the `basic_file_info.pro` file should be open. Add the line following (shown in bold in the following code):

   ```idl
   pro FILE_INFO, event
     ENVI_SELECT, title='choose a file', fid=in_fid
     IF (in_fid eq -1L) THEN return
     ENVI_FILE_QUERY, in_fid, ns=ns, nl=nl, nb=nb, fname=fname
     OpenR, unit, fname, /Get_LUN
     info = FSTAT(unit)
     Free_LUN, unit
     print, 'you selected ',fname
     print, 'number of samples = ',ns
     print, 'number of lines = ',nl
     print, 'number of bands = ',nb
     print, 'file size in bytes = ',info.size
   end
   ```

2. Save the modified user function code (overwriting the previous version in the `SAVE_ADD` directory).
Note

Remember, if a procedure has already been compiled at least one time in an IDL session, modifying and saving the procedure won’t cause IDL to use the updated version. You have to re-compile the modified version or IDL will continue to use the older version that’s already in RAM.

3. Recompile the modified version of BASIC_FILE_INFO.

4. Now, go back to the ENVI session and try running your modified user function, once again clicking “Cancel” in the file selection dialog.

   The user function now checks for the user cancelling the routine and returns.

Admittedly, our first user function did not accomplish much as far as extending ENVI’s functionality. It was designed simply to give you some practice manipulating the ENVI menu system and adding user function code.
Adding Widgets to User Functions

User functions can take virtually any form and can be used for just about any purpose. Even though user functions are selected interactively from the ENVI menu and run from within an ENVI session, they are not required to have any user interaction. For example, a user function could be used to execute batch processing. However, user functions are particularly well suited for more complicated routines that do require user input. While it is possible to collect user input from the command line, this approach is not intuitive for the ENVI user, as no other ENVI routine requires command line input. Further, because the command line is not accessible in ENVI RT (Runtime), this approach is limiting. Using widgets to interactively collect input is much easier and more straightforward for the user, and follows the existing model for ENVI routines.

Using widgets to collect input in your user functions is much easier than adding widgets to an ordinary IDL program because there is no need to write your own event handler procedures to manage the widget events—ENVI will auto-manage the widget events for you. In addition, ENVI’s special compound widgets are included in the library of ENVI routines, making it easy to create user functions that have the look and feel of ENVI. ENVI’s auto-managed widgets allow you to create user functions that fit seamlessly into ENVI.
An Illustrated Glossary of ENVI Widgets

ENVI provides more than 20 compound widgets that can be incorporated into user functions. Most of these routines begin with WIDGET_ and are listed in Chapter 9, “ENVI Routines” and in the online help. Some of the ENVI compound widgets include working buttons, such as the “Choose” button that you frequently see when selecting an output file name.

The ENVI widgets that are commonly used as building blocks for creating graphical user interfaces are illustrated here:

**ENVI_PICKFILE**

A widget used to select a filename from disk. While this is typically used to select image files, you can use this widget to collect the name of any type of file (for example, an ROI filename).

*Figure 4-2: ENVI filename selection widget.*

**ENVI_SELECT**

ENVI’s standard file selection dialog used to select an opened file, spatial and spectral subsetting, and mask bands. This widget also includes buttons that will allow
the user to open a new file from disk, open a spectral library, or restore ROIs from a file.

Figure 4-3: A standard ENVI input file selection widget.

**WIDGET_EDIT**

Provides a widget for editing items from a list.

Figure 4-4: A widget with editable items displayed in a list.
Chapter 4: User Functions

WIDGET_GEO

A widget used to prompt the user to select latitude and longitude values.

Figure 4-5: A widget for entering latitude and longitude values.

WIDGET_MAP

A compound widget used to edit map coordinates and projections.

Figure 4-6: A widget for editing map coordinates and projections.
WIDGET_MENU

A compound widget used to make a menu of exclusive buttons or non-exclusive buttons. Exclusive buttons will have round radio buttons, non-exclusive buttons will have square boxes with check marks.

Figure 4-7: A widget with exclusive radio buttons.

WIDGET_MULTI

A compound widget used to select multiple items from a list; used in ENVI to select which ROIs to use for a classification.

Figure 4-8: A widget for selecting items from a list.
WIDGET_OUTF

A widget used for selecting an output filename.

Figure 4-9: A widget for selecting or choosing an output filename.

WIDGET_OUTFM

A widget used to select an output filename. It also provides the option for the result to be saved to memory.

Figure 4-10: A widget for selecting output to file or memory. If “File” is selected, the widget is used for entering or choosing an output filename.
**WIDGET_PARAM**

A compound widget used for entering numeric parameters.

![Figure 4-11: A widget for entering numeric parameters.](image)

**WIDGET_PMENU**

A “button” pulldown menu. This compound widget is an alternative to using WIDGET_MENU with exclusive buttons.

![Figure 4-12: A widget with a “button” pulldown menu.](image)
WIDGET_RGB

A compound widget used to modify an RGB color value with the option of using other color space models.

![Figure 4-13: A widget for modifying color values.](image)

WIDGET_SLABEL

A compound widget used to display a text message with scroll bars.

![Figure 4-14: A widget for displaying text messages.](image)
WIDGET_SLIST

A compound widget used to create lists. The selected item will be listed in a separate text box.

![WIDGET_SLIST](image)

Figure 4-15: A widget for selecting items from a displayed list.

WIDGET_SSLIDER

A compound widget used to set a numeric value using a slider.

![WIDGET_SSLIDER](image)

Figure 4-16: A widget with a slider for specifying numeric values.
**WIDGET_STRING**

A compound widget used to enter strings.

![WIDGET_STRING](image)

*Figure 4-17: A widget for entering text.*

**WIDGET_SUBSET**

ENVI’s standard widget for specifying spatial subsets. This widget will produce a button labeled “Spatial Subset,” which will bring up the standard spatial subset dialog.

![WIDGET_SUBSET](image)

*Figure 4-18: A standard spatial subset widget.*

**WIDGET_TOGGLE**

A compound widget used to create a toggle button with ENVI’s arrow icon.

![WIDGET_TOGGLE](image)

*Figure 4-19: A widget with an arrow toggle button.*
Auto-Managed Widget Events

Ordinarily, programs that include widgets must have separate procedures to handle all of the events that the widgets might generate. These event handler procedures can be cumbersome and often confusing to code, especially for a relatively new programmer. To facilitate adding widgets to ENVI user functions, the developers have provided a mechanism that allows ENVI to automatically manage all of the events that originate from ENVI widgets. There are two ENVI routines that are central to being able to use auto-managed widgets.

WIDGET_AUTO_BASE

In ordinary IDL Widget programming, all Widget Bases, including the Top Level Base (or TLB), are created using the WIDGET_BASE function. However, in ENVI programming, if you want to create widget hierarchies whose events are auto-managed by ENVI, you must use the special ENVI function called WIDGET_AUTO_BASE to create the TLB. All other bases used in building the GUI for the user function should be created with the ordinary WIDGET_BASE function.

Top Level Bases created using WIDGET_AUTO_BASE are automatically column-based, centered, and modal (i.e., blocking). Unlike WIDGET_BASE, which can accept dozens of different keywords that will control the appearance of the base, WIDGET_AUTO_BASE only accepts four keywords: GROUP, TITLE, XBIG, and YBIG. These keywords allow the GUI to be tied to an existing widget, to be given a title for the window bar, and to automatically be offset if unusually wide or tall.

AUTO_WID_MNG

In an ordinary IDL widget program, once the GUI is defined, the XMANAGER procedure is called to register the widget and to begin monitoring for events (that the programmer would have to manage with a separate event handler procedure). However, for auto managed widgets the ENVI programmer never needs to call XMANAGER. Instead, a special ENVI routine called AUTO_WID_MNG is called which internally registers the widget, monitors for events, and returns the user input to the programmer in the form of a structure variable. The AUTO_WID_MNG function will also add two buttons labeled “OK” and “Cancel” to the bottom of the GUI.

This function is used as follows:

```
result = AUTO_WID_MNG(TLB)
```
The argument of the function (TLB) must be a widget base that was created with WIDGET_AUTO_BASE. When the user function’s GUI is dismissed (for example, by the user clicking on the “OK” button to proceed with the routine), RESULT is set to a structure variable that contains a special tag for each widget that was used in the GUI. The name of each tag in the structure is defined by the user value (i.e., the UVALUE keyword) assigned to each ENVI compound widget when the GUI was defined.

This technique lets ENVI user functions take a very simple form:

```pro
pro MY_USER_FUNCTION, Event
    TLB = WIDGET_AUTO_BASE(...)    
    lst_parameter = WIDGET_PARAM(TLB, uvalue='param1'...) 
    result = AUTO_WID_MNG(TLB)     
    do the processing...
end
```

In this example, the value of result.param1 would contain the information that the user entered into the WIDGET_PARAM widget. The type of information that is returned from an ENVI widget is described in the reference page for the specific routine. In addition to tags for each ENVI widget that was used in the GUI, the RESULT structure will also always contain a tag called ACCEPT which is set to 1 if the “OK” button was selected and zero if the “Cancel” button was selected.

Of course, if you prefer, you can choose to write your own event handler procedures for your user functions. If you choose to do this, you can still use the ENVI compound widgets in your code. The ENVI widget will simply produce an event structure with an extra tag called RESULT that will contain the information entered into the widget. There are indeed some cases when it is preferable to write your own event handlers, for example if you need to create a user function whose GUI is not modal; however, the vast majority of user functions can be written without this additional complication.

**Example: Building a Simple GUI with Auto-Managed Widgets**

The purpose of this example is to familiarize you with building a graphical user interface and with working with the result structure. This user function will simply collect two numeric parameters and prompt the user if they should be added or multiplied.

We will call this user function TEST_WIDGETS.

1. Open the envi.men file and add an item under the MyFuntions menu:

   ```
   1 {Test Widgets} {not used} {test_widgets}
   ```
Note — Remember to save the menu file.

2. Restart ENVI. In the IDLDE window that is running the ENVI session, open a new editor window.

3. In the new editor window, enter the following user function code:

```idl
pro TEST_WIDGETS, event

compile_opt STRICTARR

TLB = WIDGET_AUTO_BASE(title='widget test')

p1 = WIDGET_PARAM(tlb, /auto_manage, dt=4, field=2, prompt='enter the first parameter', uvalue='p1')

p2 = WIDGET_PARAM(tlb, /auto_manage, dt=4, field=2, prompt='enter the second parameter', uvalue='p2')

operation = WIDGET_TOGGLE(tlb, /auto_manage, default=0, list=['add', 'multiply'], prompt='operation', uvalue='operation')

result = AUTO_WID_MNG(TLB)
IF (result.accept eq 0) THEN return

IF (result.operation eq 0) THEN $ ENVI_INFO_WID, STRTRIM(result.p1 + result.p2) ELSE $ ENVI_INFO_WID, STRTRIM(result.p1 * result.p2)

end
```

4. Save the file in the SAVE_ADD directory as test_widgets.pro.

5. Compile the procedure to make sure there were no syntax errors.

6. After TEST_WIDGETS is compiled, place your cursor on the line towards the bottom of the program that begins with

```idl
IF (result.operation eq 0) THEN $
```

7. Then, from the Run menu in the IDLDE select Set Breakpoint.

You should see a yellow circle appear next to this line in the Editor window. Setting a breakpoint will allow you to execute the user function, but will stop at the line where the breakpoint occurs. This will allow you to examine some of the variables that are created within the TEST_WIDGETS procedure.
8. After the breakpoint is set, try running the TEST_WIDGETS user function. You can execute the function either by using the new menu item under MyFunctions, or using the “Run” button on the Editor’s toolbar.

9. When the widget interface appears, pay particular attention to the layout of the widgets.

For example, examine the widget to see if the toggle button appears on one line or two, whether the widget GUI looks the same as ENVI’s GUIs usually look, and to see if the widget interface could be organized better.

It takes some practice to control the layout of any widget interface. It is usually possible to force widgets to appear on a single line by putting them into their own widget base that is defined as a row base instead of a column base.

10. After you fill in values and select an operation, click on the “OK” button. The user function should now halt execution because of the breakpoint.

11. In the IDLDE use the Variable Watch Window to examine the RESULT variable, or use the HELP command:

   ```IDL
   ENVI> help, result, /struct
   ** Structure <bed2b8>, 4 tags, length=24, refs=1:
   P1            DOUBLE       32.000000
   P2            DOUBLE       24.800000
   OPERATION     INT          1
   ACCEPT        INT          1
   ```

12. The result structure contains four tags, one for each of the two WIDGET_PARAMS (P1 and P2), one for the WIDGET_TOGGLE (OPERATION), and one to indicate which of the “OK” and “Cancel” buttons was selected (ACCEPT). Note that the names of the tags are the same as the user values for the widgets. The ACCEPT tag is always automatically added to the result structure.

   The value RESULT.P1 and RESULT.P2 are set to the entered parameters, RESULT.OPERATION is set to the index into the LIST array that corresponds to the value selected (0 for add and 1 for multiply), and RESULT.ACCEPT is set to 1 to indicate that the “OK” button was selected. If the ACCEPT tag is set to zero then the user selected the “Cancel” button (if this occurs then the user function will return in the next line).

   Also note that the numeric data returned by AUTO_WID_MNG are in double precision. This will always be the case for numeric data collected by ENVI’s AUTO_WID_MNG routine, so in some cases you may need to change the data type back to something more appropriate for your needs.
13. From the IDLDE’s *Run* menu, select *Step Out* to continue execution of the user function.

The results of the operation are displayed in an ENVI report widget created with the function `ENVI_INFO_WID`. This is a special widget routine that makes a simple dialog window in which to display text reports. This widget is not modal – that is, the widget does not block the rest of ENVI from functioning.

14. With a few minor modifications, we can significantly improve the appearance of the widget interface. Make the following changes to the `TEST_WIDGETS` routine (shown in bold in the following code).

```idl
pro TEST_WIDGETS, event

compile_opt STRICTARR

TLB = WIDGET_AUTO_BASE(title='widget test')

row_base1 = WIDGET_BASE(TLB, /row)
p1 = WIDGET_PARAM(row_base1, /auto_manage, dt=4, $ field=2, prompt='enter the first parameter', $ uvalue='p1')

row_base2 = WIDGET_BASE(TLB, /row)
p2 = WIDGET_PARAM(row_base2, /auto_manage, dt=4, $ field=2, prompt='enter the second parameter', $ uvalue='p2')

row_base3 = WIDGET_BASE(TLB, /row)
operation = WIDGET_TOGGLE(row_base3, /auto_manage, $ default=0, list=['add', 'multiply'], $ prompt='operation', uvalue='operation')
result=AUTO_WID_MNG(TLB)

IF (result.operation eq 0) THEN $
ENVI_INFO_WID, STRTRIM(result.p1 + result.p2) ELSE $
ENVI_INFO_WID, STRTRIM(result.p1 * result.p2)
end
```

15. Save the user function code (overwriting the original version in the SAVE_ADD directory), and compile it to be sure there are no syntax errors.

16. Place the cursor on the line containing the breakpoint, then from the *Run* menu select *Clean All Breakpoints*.

17. Now, try running the modified version of the `TEST_WIDGETS` user function.
Note that the widget layout has changed so that all of the individual widgets now appear on only one line.
Trapping Errors in User Functions

Whenever possible, it is strongly recommended that any potential errors that could occur during the execution of a user function be trapped internally to prevent ENVI from crashing. While it is impossible (and unreasonably time consuming) to account for every possibility, there are several quick and easy ways of preventing the most common crashes. General guidelines include:

- When a widget includes a “Cancel” button, always check to see if it was selected (every ENVI_SELECT, ENVI_PICKFILE, and AUTO_WID_MNG widget has a “Cancel” button). This type of error checking is very easy to implement and should always be included in a user function.

- Wherever possible, hard code default values into widgets or algorithms in case the user leaves a required value undefined.

- When using the WHERE function, always use the optional COUNT parameter to make sure a valid result was returned.

Input/Output Error Handling

Although not a requirement, it is recommended that I/O errors are properly handled in all processing routines. This is accomplished using the IDL ON_IOERROR procedure which specifies a “jump to” statement in the event of an I/O error in the current procedure. If ON_IOERROR is called, and an I/O related error later occurs in the same procedure activation, control is transferred to the designated statement with the error code stored in the system variable !ERROR_STATE. The text of the error message is contained in !ERROR_STATE.MSG.

In addition to ON_IOERROR is the CATCH procedure, which provides a more generalized mechanism for the handling of exceptions and errors. The advantage of CATCH is that it will trap not only the I/O errors by also any programming errors such as undefined variable, invalid array subscripts, or undefined functions.

ENVI uses a mix of ON_IOERROR and CATCH for its error handling. Processing routines internally use ON_IOERROR with a CATCH mechanism setup prior to the processing routine call. Any I/O errors are handled within the routine but other programming errors are handled externally. In order to follow this model the event handler would set up a CATCH mechanism after getting the processing parameters and prior to calling the processing routine.

ENVI provides the procedure ENVI_IO_ERROR for I/O error display and to optionally delete the output file being created. The use of this routine keeps the user function I/O error display consistent with ENVI.
Chapter 4: User Functions

The error handling routines used in ENVI are listed in the following table.

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATCH</td>
<td>Error trapping</td>
<td>247</td>
</tr>
<tr>
<td>ENVI_IO_ERROR</td>
<td>Error reporting</td>
<td>454</td>
</tr>
<tr>
<td>ON_IOERROR</td>
<td>I/O Error trapping</td>
<td>605</td>
</tr>
</tbody>
</table>

Table 4-1: Error handling routines available in ENVI.

The following example summarizes the use of the error handling routines.

**Example: Input/Output Error Handling**

This example illustrates the use of ON_IOERROR and ENVI_IO_ERROR (see “ENVI_IO_ERROR” on page 454) working together to trap and display processing routine I/O errors. It is recommended that all processing routines handle I/O errors and display an appropriate error message. The following steps outline a technique for handling I/O errors in the processing routine.

1. At the start of the routine, clear the system error code, !ERROR_STATE using the MESSAGE routine, and declare the jump location for an I/O error.
2. When the processing is completed, clear the system error code, !ERROR_STATE using the MESSAGE routine, to remove any non-fatal errors.
3. Determine if there was an I/O error and print the error message.
4. Delete the current output file specified by the file unit number UNIT.

The following model can be used for handling I/O errors.

```pro
pro user_function, [parameters and keywords]
message,/reset
on_ioerror, trouble
 Initialization and Processing ...
message,/reset
trouble: if (!error_state.code ne 0) then $
 env_i_o_error, 'User Function', unit=unit
 if (!error_state.code eq 0) then $
 Write an ENVI header
end
```
This model is only a suggested method to deal with I/O errors. Many other effective techniques are available.

**Using CATCH for Unexpected Non-Input/Output Errors**

CATCH is a special IDL procedure that allows you to use IDL’s internal error catching mechanism to make a generic error handler that will prevent a crash when an unexpected error condition occurs. The IDL system variable !ERROR_STATE contains information about the last error, including the internal error code (!ERROR_STATE.CODE) and the text of the error message (!ERROR_STATE.MSG). Each time an IDL error occurs, this system variable is updated. CATCH’s usage is quite simple:

```
Catch, error
```

When CATCH is called, the ERROR variable is set to zero. However, when an error condition occurs, ERROR is set to the internal error code, !ERROR_STATE.CODE, and the IDL procedure that is executing (i.e., the ENVI user function) jumps immediately to the line of the program where the CATCH procedure was called and continues executing from there. At this point, you can use the information in the !ERROR_STATE system variable to display a message about the error and prompt the user to choose how they wish to proceed.

For example, the following code excerpt works well as a generic error handler when placed at the beginning of a user function:

```
Catch, error
If (error NE 0) Then Begin
   ok = DIALOG_MESSAGE(!error_state.msg, /cancel)
   If (STRUCASE(ok) EQ 'CANCEL') Then return
Endif
```

```
Working With Image Displays

Each three-window image display group in ENVI can be identified with a unique Display Number, or DN. Once a DN for a particular display is obtained, there are several ENVI routines that can provide very useful information about the image data that are displayed, as well as controls for moving the position of the Zoom window.

**Note**

For complete descriptions and example code for these routines, see Chapter 9, “ENVI Routines”.

**ENVI_DISP_QUERY**

This routine provides fundamental information about the image displayed in the window, including the FID of the image file, the spatial dimensions of the file, the type of image displayed (RGB, grayscale, or classification), the band positions that are displayed, and the size (in pixels) of the three display windows.

**ENVI_GET_IMAGE**

This routine is equivalent to ENVI_GET_DATA except that it returns data displayed in the image window instead of that stored in an image file. Given the band positions, dimensions, and DN, ENVI_GET_IMAGE will return the bytescale stretched data.

**DISP_GET_LOCATION**

This routine will return the location of the currently selected pixel (i.e., the pixel on which the Zoom window is centered).

**DISP_GOTO**

This routine will move the Zoom window to a specified location, updating the Image and Scroll windows if necessary.
Using Processing Routines and Tiling

Processing routines in ENVI take input image data, process the data and output a new image, plot, or report. ENVI commonly refers to these as the “doit” portion of a user function.

ENVI processing routines are usually combined with ENVI’s tiling methodology to handle images of any spatial or spectral size. Tiles can be spatial or spectral with the size of a spatial tile defined in the configuration file. The size of a spectral tile is always the number of samples times the number of bands.

All ENVI user functions can use the built-in tiling functions to get data. This ensures that the processing function will operate on any size data file, both spatially and spectrally. ENVI tiles come in three formats; BSQ format (band sequential: number of samples by number of tile lines), BIL format (band interleaved by line: number of samples by number of bands), or BIP format (band interleaved by pixel: number of bands by number of samples). BIL and BIP tiles differ only by a transpose:

\[ \text{BIL tile} = \text{transpose(BIP tile)} \]

ENVI also provides status window functions that summarize tile processing status, see “Processing Status Report” on page 117.

ENVI also provides for non-tiled processing, however, this is not a recommended solution because it does not always work for large images. Nevertheless, non-tiled processing is a quick way to access data without the overhead of setting up tiling operations and can be used as a prototyping technique (see “Non-Tiled Processing Routines” on page 115).

All processing functions should include both error handling and status reports as common practice. The next sections break down the processing routine into its basic components and discuss the available ENVI routines.

**Note**

Creating the processing routine as a separate procedure from the menu event handler and processing parameter input allows the routine to be called directly with all supplied arguments. ENVI refers to this as Batch Mode. See “Batch Mode ENVI” on page 53.

**Tiled Processing Routines**

“ENVI’s tiling process divides input data into equal size units, either spatially or spectrally, ensuring that all images regardless of spatial or spectral size can be processed. A spatial tile is a group of lines with all samples on that line while a
spectral tile is one line of all bands. Spatial and spectral tiles are illustrated in Figure 4-20.

Figure 4-20: Illustration of spatial and spectral tiles.

Spatial tiles are equal or near equal divisions of the input band and are intended to be used for spatial processing regardless of the file storage order. When accessing multiple bands from a single file, each band will have an equivalent number of spatial tiles. For example, the first tile from bands 1 and 2 will have the same number of lines.

It is not uncommon for a processing routine to use spatial tiles for BSQ files and spectral tiles for BIL or BIP files. Using tiles with the same storage order as the input is most efficient. For example, when applying a gain and offset to all the bands in an image, it is much more efficient to operate spatially on data stored as BSQ and operate spectrally on data stored as BIL or BIP.

Spatial tiles can also specify the number of overlap lines when performing neighborhood operations with tiling, thus eliminating the need to keep information between tiles. The overlap is only added to the top of each tile and is not used when a single band is returned as one tile, the case where the number of tiles equals the number of input bands. For example, a 3x3 convolution would use a tile overlap to allowing processing of the bottom line of the previous tile.

Note

The input data format, BSQ, BIL or BIP, is returned using the INTERLEAVE keyword to ENVI_FILE_QUERY.
The steps for a tiling process are:

1. Initialize the spatial or spectral tile request, `ENVI_INIT_TILE`.
2. Retrieve a tile of input data, `ENVI_GET_TILE`.
3. Free the tile request when no more tiles needed, `ENVI_TILE_DONE`.

The following examples will detail the use of the tiling routines and issues related to the input file storage order. The following table lists the routines used to initialize tiles, get tile data, and free the tile request.

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ENVI_INIT_TILE</code></td>
<td>Initialize a tile request</td>
<td>452</td>
</tr>
<tr>
<td><code>ENVI_GET_TILE</code></td>
<td>Get a tile of data</td>
<td>446</td>
</tr>
<tr>
<td><code>ENVI_TILE_DONE</code></td>
<td>Free the tile request</td>
<td>532</td>
</tr>
</tbody>
</table>

**Table 4-2: ENVI tiling routines.**

**Example: Spatial Tiling**

Tiling routines processing only spatial tiles do not need to worry about the input data storage order. Tiles are initialized as spatial tiles and each tile is processed in the same way. The tile initialization routine needs the file id, the selected bands, and the spatial dimensions. `ENVI_SELECT` (see “ENVI_SELECT” on page 500) returns all of these parameters when used to select the input file.

The procedure in this example uses `ENVI_SELECT` to choose the input file, returning the FID, POS and DIMS variables as inputs to `ENVI_INIT_TILE` (see “ENVI_INIT_TILE” on page 452). For spatial tiles the interleave value is 0, BSQ, regardless of the data interleave.

**Note**

The values zero, one, and two correspond to BSQ, BIL, and BIP interleave, respectively.

The returned TILE_ID from `ENVI_INIT_TILE` is a unique reference id associated with the requested tiles and is used by the other tile routines. A FOR loop is then used to loop over the total number of tiles, the NUM_TILES returned from the tile initialization. Within the FOR loop, the current tile, band index, and tile data information are printed to the IDL command log window. Once all the tiles have been
retrieved the tile request is freed using ENVI_TILE_DONE (see “ENVI_TILE_DONE” on page 532).

The sample code listing follows and can be found in the file uttile1.pro in the lib directory of the installation.

```idl
pro spat_tile
  envi_select, title='Input Filename', fid=fid, $ pos=pos, dims=dims
  if (fid eq -1) then return
  tile_id = envi_init_tile(fid, pos, interleave=0, $ num_tiles=num_tiles, xs=dims[1], xe=dims[2], $ ys=dims[3], ye=dims[4])
  for i=0L, num_tiles-1 do begin
    data = envi_get_tile(tile_id, i, band_index=band_index)
    print, i, band_index
    help, data
  endfor
  envi_tile_done, tile_id
end
```

As shown by the example, all spatial tiling is performed in the same way, regardless of the data storage order. To execute this example:

1. Save the routine to a file and place in the save_add directory.
2. Start or restart ENVI.
3. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).
4. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to create the Status dialog.
   ```idl
   spat_tile
   ```
5. Choose an input file and look at the messages displayed in the IDL command log window.

---

**Example: Spectral Tiles**

Tiling routines processing only spectral tiles need to worry about BIL and BIP tile interleaves for optimal performance. The tile initialization routine needs the file id,
selected bands, and spatial dimensions. ENVI_SELECT (see “ENVI_SELECT” on page 500) returns all of these parameters when used to select the input file.

**Note**

If performance is not an issue, then for simplicity always use BIL tiles.

The procedure in this example uses ENVI_SELECT to choose the input file, returning the selected FID, POS and DIMS variables for inputs to ENVI_INIT_TILE (see “ENVI_INIT_TILE” on page 452). The spectral tiles in this example use BIL tiles for data in BSQ or BIL interleave and BIP tiles for data in BIP interleave. The input data interleave format is returned using the INTERLEAVE keyword to ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411).

**Note**

The values zero, one, and two correspond to BSQ, BIL, and BIP interleave, respectively.

The returned TILE_ID from ENVI_INIT_TILE is a unique reference id associated with the requested tiles and is used by the other tile routines. A FOR loop is then used to loop over the total number of tiles, the NUM_TILES returned from the tile initialization. Within the FOR loop the current tile and tile data information are printed to the IDL command log window. Once all the tiles have been retrieved, the tile request is freed using ENVI_TILE_DONE (see “ENVI_TILE_DONE” on page 532).

The sample code listing follows and can be found in the file uftile2.pro in the lib directory of the installation.

```idl
pro spec_tile
  envi_select, title='Input Filename', fid=fid, $ pos=pos, dims=dims
  if (fid eq -1) then return
  envi_file_query, fid, interleave=interleave
  tile_id = envi_init_tile(fid, pos, num_tiles=num_tiles, $ interleave=(interleave > 1), xs=dims[1], xe=dims[2], $ ys=dims[3], ye=dims[4])
  for i=0L, num_tiles-1 do begin
    data = envi_get_tile(tile_id, i)
    print, i
    help, data
  endfor
  envi_tile_done, tile_id
end
```
To execute this example:

1. Save the routine to a file and place in the `save_add` directory.
2. Start or restart ENVI.
3. Open the IDL development environment (PC, Macintosh) or the shell window if ENVI was started in (UNIX).
4. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to create the Status dialog.
   ```idl
   spec_tile
   ```
5. Choose an input file and look at the messages displayed in the IDL command log window.

**Note**
The greater than operator (>) used in the call to ENVI_INIT_TILE will return the greater of its two arguments. For details, see the *IDL Reference Guide*.

**Example: Spatial and Spectral Tiles**

Tiling routines which can process using either spatial or spectral tiles should set the tile interleave equal to the file interleave. For example, a gain and offset correction applied to every band in an image could be applied using any of the file interleaves. With a BSQ interleave, the current gain and offset values would be based on the band index of the current tile. BIL and BIP tile interleaves would apply gains and offsets as a vector to every band in the current tile. The tile initialization routine needs the file id, selected bands, and spatial dimensions. ENVI_SELECT (see “ENVI_SELECT” on page 500) returns all of these parameters when used to select the input file and is used in this example. For details see “ENVI_INIT_TILE” on page 452. The input image’s interleave is found using the INTERLEAVE keyword to ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411).

**Note**
The values zero, one, and two correspond to BSQ, BIL, and BIP interleave, respectively.

The returned TILE_ID from ENVI_INIT_TILE is a unique reference id associated with the requested tiles and is used by the other tile routines. A FOR loop is then used to loop over the total number of tiles, the NUM_TILES returned from the tile initialization. Within the FOR loop the current tile and tile data information are printed to the IDL command log window. Typically, within the FOR loop is a case statement on the interleave type to allow processing of each tile format. A sample
case statement has been included in the example. Once all the tiles have been retrieved, the tile request is freed using ENVI_TILE_DONE (see “ENVI_TILE_DONE” on page 532).

The sample code listing follows and can be found in the file uftile3.pro in the lib directory of the installation.

```idl
pro ss_tile
  envi_select, title='Input Filename', fid=fid, $ pos=pos, dims=dims if (fid eq -1) then return envi_file_query, fid, interleave=interleave tile_id = envi_init_tile(fid, pos, num_tiles=num_tiles, $ interleave=interleave, xs=dims[1], xe=dims[2], $ ys=dims[3], ye=dims[4]) for i=0L, num_tiles-1 do begin data = envi_get_tile(tile_id, i) ; Sample case statement for BSQ, BIL and BIP tiles case interleave of 0: 1: 2: endcase print, i help, data endfor envi_tile_done, tile_id end
```

To execute this example.

1. Save the routine to a file and place in the save_add directory.
2. Start or restart ENVI.
3. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).
4. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to create the Status dialog.
   ```idl
   ss_tile
   ```
5. Choose an input file and look at the messages displayed in the IDL command log window.
**Saving the Results**

This section will discuss how to save image results from tiled processing. For information on outputting results to plots, see “Plotting” on page 125, and for information on outputting results to a report, see “Reports” on page 127. Image results can be written to a file or saved in memory; ENVI allows both options whenever possible. When writing results to a file it is recommended that the result is written with the same interleave as the tile processing. Tiles processed using spatial BSQ tiles would generate BSQ results and similarly BIL tiles would generate BIL results. Following this simplification, the processed tiles are written directly to files without the need to convert interleave formats.

Output files are opened for writing using the IDL routine OPENW. Prior to calling OPENW, a file unit number is allocated using GET_LUN. The tile data are written to the file using the IDL routine WRITEU which writes the specified array of data in binary format. After all the data are written to the file, the file is closed and the file unit number is de-allocated using the IDL routine FREE_LUN.

Once the file is written to disk, the ENVI routine ENVI_SETUP_HEAD (see “ENVI_SETUP_HEAD” on page 510) is used to write an ENVI header and optionally open the file. The following file information is required to write the header; filename, samples, lines, bands, offset, interleave, and data type. Although optional, the following information should also be supplied X and Y starting pixel, text description, band names, and inheritance. The optional OPEN keyword opens the file and places the contents into the Available Bands List. Also, remember to use the WRITE keyword to write the header file to disk, otherwise it is stored in memory only and will be lost when the IDL session ends.

**Note**

Additional information is required for special file types, such as classification and spectral libraries, see “ENVI_SETUP_HEAD” on page 510 for more information.

For memory output, the result is stored in an allocated array in memory. The processed data from each tile are inserted into the appropriate storage location. The dimensions of a memory array are always NS x NL x NB. The IDL functions BYTARR, INTARR, LONARR, FLTARR, DBLARR, and MAKE_ARRAY are used to create memory arrays of type byte, integer, long, floating-point, double-precision floating-point, and arbitrary, respectively.

When the processing is completed and the memory array contains the processed result, the array is passed into ENVI using the routine ENVI_ENTER_DATA (see “ENVI_ENTER_DATA” on page 389). At a minimum, only the memory array is
required. Although optional, the following information should also be supplied: X and Y starting pixels, text description, band names, and inheritance.

The following examples will detail the use of writing header files and entering memory results.

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_ENTER_DATA</td>
<td>Enter a memory array into ENVI</td>
<td>389</td>
</tr>
<tr>
<td>ENVI_SETUP_HEAD</td>
<td>Write an ENVI header file</td>
<td>510</td>
</tr>
</tbody>
</table>

Table 4-3: Routines for entering results into ENVI.

**Example: Saving Spatial Tiles to Disk**

The “Example: Spatial Tiling” on page 104 was modified to save the resulting tiles to disk and an ENVI header file is written. The new modifications are:

1. Accept an output file name as a parameter
2. Open the output file and write the result
3. Call ENVI_SETUP_HEAD to write an ENVI header file.

The IDL procedure OPENW is used to open the output filename and the keyword GET_LUN allocates a file unit number. Within the tile processing loop, each returned data tile is written to disk using the IDL procedure WRITEU. WRITEU uses the file unit number returned to from OPENW and the tile data array as arguments. The tile data are written in binary format using the data type of the tile data array. Since no actual processing is performed, the tile data type remains the same as the input data type, the value returned from ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411) using the DATA_TYPE keyword.

**Note**

Tile processing may change the data type; make sure to check that the output data type is what you expect.

After writing all data to the output file, close the file and free the allocated file unit number using the IDL procedure FREE_LUN. The ENVI header file is written using ENVI_SETUP_HEAD (see “ENVI_SETUP_HEAD” on page 510). This example specifies both the required keywords, FNAME, NS, NL, NB, DATA_TYPE, OFFSET and INTERLEAVE, and the optional keywords, XSTART, YSTART, and DESCRIP.
The sample code listing follows and can be found in the file \texttt{uftile4.pro} in the \texttt{lib} directory of the installation.

```idl
pro spat_disk, out_name
  ; Check for an output filename
  if (n_elements(out_name) eq 0) then begin
    print, 'Please specify a valid output filename'
    return
  endif
  envi_select, title='Input Filename', fid=fid, $ pos=pos, dims=dims
  if (fid eq -1) then return
  envi_file_query, fid, data_type=data_type, xstart=xstart, $ ystart=ystart
  nb = n_elements(pos)
  openw, unit, out_name, /get_lun
  tile_id = envi_init_tile(fid, pos, interleave=0, $ num_tiles=num_tiles, xs=dims[1], xe=dims[2], $ ye=dims[3], ye=dims[4])
  for i=0L, num_tiles-1 do begin
    data = envi_get_tile(tile_id, i, band_index=band_index)
    print, i, band_index
    writeu, unit, data
  endfor
  free_lun, unit
  envi_setup_head, fname=out_name, ns=ns, nl=nl, nb=nb, $ data_type=data_type, offset=0, interleave=0, $ xstart=xstart+dims[1], ystart=ystart+dims[3], $ descirp='Test routine output', /write, /open
  envi_tile_done, tile_id
end
```

To execute this example:

1. Save the procedure to a file and place in the \texttt{save_add} directory.
2. Start or restart ENVI.
3. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).
4. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to save the tiles to disk, where \textit{filename} is a valid output file name.

```
spat_disk, 'filename'
```
5. Choose an input file.
6. The resulting image is added to the Available Bands List.

**Example: Saving Spatial Tiles to Memory**

The “Example: Saving Spatial Tiles to Disk” on page 110 was modified to save the resulting tiles to memory. The data are then entered into ENVI using ENVI_ENTER_DATA (see “ENVI_ENTER_DATA” on page 389).

The IDL function MAKE_ARRAY is used to allocate an output memory array for the tiled data. In this example, the output array is the same data type as the input so the TYPE keyword to MAKE_ARRAY is set to the data type returned form ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411).

**Note**

The output data type is not always the same as the input data type. It depends upon the processing routine and the range of the output data.

To save each tile into the output array, the band index and Y start value returned from ENVI_GET_TILE (see “ENVI_GET_TILE” on page 446) are used to index into the appropriate array location. The Y start returned from ENVI_GET_TILE is in file coordinates and any spatial line subset, DIMS[3], must be subtracted. After saving all the tile data into the output memory array, the data are entered into ENVI using ENVI_ENTER_DATA. This example specifies the optional keywords, XSTART, YSTART, and DESCRIP to ENVI_ENTER_DATA.

The sample code listing follows and can be found in the file utile5.pro in the lib directory of the installation.

```
pro spat_mem
  envi_select, title='Input Filename', fid=fid, $
      pos=pos, dims=dims
  if (fid eq -1) then return
  envi_file_query, fid, data_type=data_type, xstart=xstart,$
      ystart=ystart
  nb = n_elements(pos)
  mem_res = make_array(ns, nl, nb, type=data_type, /nozero)
  tile_id = envi_init_tile(fid, pos, interleave=0, $
      num_tiles=num_tiles, xs=dims[1], xe=dims[2], $
      ys=dims[3], ye=dims[4])
  for i=0L, num_tiles-1 do begin
    data = envi_get_tile(tile_id, i, band_index=band_index, $
        ys=ys)
    print, i, band_index
    mem_res[0,ys-dims[3],band_index] = data
  endfor
```
To execute this example:

1. Save the procedure to a file and place in the `save_add` directory.
2. Start or restart ENVI.
3. Open the IDL development environment (PC, Macintosh) or the shell window in which ENVI was started (UNIX).
4. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”): `spat_mem`.
5. Choose an input file.
6. The resulting image is added to the Available Bands List.

**Example: Saving Spectral Tiles to Disk**

The “Example: Saving Spatial Tiles to Disk” on page 110 was modified to save spectral tiles to disk.

The IDL procedure OPENW is used to open the output filename and the keyword GET_LUN allocates a file unit number. Within the tile processing loop, each returned data tile is written to disk using the IDL procedure WRITEU. WRITEU uses the file unit number returned from OPENW and the tile data array as arguments. The tile data are written in binary format using the data type of the tile data array. Since no actual processing is performed, the tile data type remains the same as the input data type, the value returned from ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411) using the DATA_TYPE keyword.

**Note**

Tile processing may change the data type, make sure to check that output data type is what you expected.

Writing each tile in the data tile interleave also preserves the file interleave. Both the input and output file have the same interleave. The input file interleave is returned from ENVI_FILE_QUERY using the INTERLEAVE keyword.

After writing all data to the output file, close the file and free the allocated file unit number using the IDL procedure FREE_LUN. The ENVI header file is written using ENVI_SETUP_HEAD. This example specifies both the required keywords,
FNAME, NS, NL, NB, DATA_TYPE, OFFSET and INTERLEAVE, and the optional keywords, XSTART, YSTART, and DESCRIP.

The sample code listing follows and can be found in the file {	t uftile6.pro} in the lib directory of the installation.

```idl
pro spec_disk, out_name
  ; Check for an output filename
  if (n_elements(out_name) eq 0) then begin
    print, 'Please specify a valid output filename'
    return
  endif
  envi_select, title='Input Filename', fid=fid, $
    pos=pos, dims=dims
  if (fid eq -1) then return
  envi_file_query, fid, data_type=data_type, xstart=xstart,$
    ystart=ystart, interleave=interleave
  nb = n_elements(pos)
  openw, unit, out_name, /get_lun
  tile_id = envi_init_tile(fid, pos, num_tiles=num_tiles, $
    interleave=(interleave > 1), xs=dims[1], xe=dims[2], $
    ys=dims[3], ye=dims[4])
  for i=0L, num_tiles-1 do begin
    data = envi_get_tile(tile_id, i)
    writeu, unit, data
    print, i
  endfor
  free_lun, unit
  envi_setup_head, fname=out_name, ns=ns, nl=nl, nb=nb, $
    data_type=data_type, offset=0, interleave=(interleave > 1),$ 
    xstart=xstart+dims[1], ystart=ystart+dims[3], $ 
    descrip='Test routine output', /write, /open
  envi_tile_done, tile_id
end
```

To execute this example:

1. Save the procedure to a file and place in the save_add directory.
2. Start or restart ENVI.
3. Open the IDL development environment (PC, Macintosh) or the shell window in which ENVI was started (UNIX).
4. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to save the tiles to disk where {	t filename} is a valid output file name.

   ```idl
   spec_disk, 'filename'
   ```
5. Choose an input file.

6. The resulting image is saved to disk and added to the Available Bands List.

Non-Tiled Processing Routines

ENVI also provides a method of performing non-tiled processing. For tiling, the input data are divided into equal size units, either spatially or spectrally. Non-tiled processing is able to access an entire spatial band, or any spatial subset, with a single request. ENVI does not put any constraints on the size of the requested data.

Spectral non-tiled processing has two options: build the whole image cube into memory using the single band requests, or request the data one spectral slice at a time. Both techniques are available without the need to initialize tiling.

Warning

If ENVI cannot allocate an array to hold the requested data, no data are returned.

Non-tile processing routines are useful to prototype algorithms on data that fit into memory. When a more general solution is desired, the non-tiled processing routines can be converted to tiled routines.

The two routines used to access non-tiled data are listed in the following table.

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_GET_DATA</td>
<td>Get a spatial piece of data</td>
<td>423</td>
</tr>
<tr>
<td>ENVI_GET_SLICE</td>
<td>Get a spectral slice of data</td>
<td>441</td>
</tr>
</tbody>
</table>

Table 4-4: Non-tiled processing routines available in ENVI.

The following examples illustrate the use of the routines in Table 4-4.

Example: Non-tiled Spatial Processing

This example interactively requests a band of spatial data. The routine ENVI_SELECT (see “ENVI_SELECT” on page 500) is used to choose the input data.

1. Start ENVI.
2. Open an multiple band image file.
3. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).

4. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to select a file.

   ```
   envi_select, title='Input Filename', fid=fid, pos=pos, $
   dims=dims
   ```

5. Type the following on the ENVI command line to return the first band of data.

   ```
   data = envi_get_data(fid=fid, dims=dims, pos=pos[0])
   ```

6. Verify the return of the data using the help command.

   ```
   help, data
   ```

7. If the number of elements of POS is greater than one then type the following on the ENVI command line to return the second band of data.

   ```
   data = envi_get_data(fid=fid, dims=dims, pos=pos[1])
   ```

This interactive example is for demonstration purposes only. In practice, these steps are part of the processing routine.

**Example: Non-tiled Spectral Processing**

This example interactively requests a spectral data slice, which is a single line of all bands. Both BIL (ns, nb) and BIP (nb, ns) slices can be requested. The routine ENVI_SELECT (see “ENVI_SELECT” on page 500) is used to choose the band of data to request.

1. Start ENVI.

2. Open a multiple band image file.

3. Open the IDL development environment (PC) or the shell window in which ENVI was started (UNIX).

4. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to select a file (.).

   ```
   envi_select, title='Input Filename', fid=fid, pos=pos, $
   dims=dims
   ```

5. Type the following on the ENVI command line to get a BIL slice of data for the first line.

   ```
   data = envi_get_slice(fid=fid, pos=pos, line=dims[3], $
   xs=dims[1], xe=dims[2], /bil)
   ```
6. Verify the return of the data using the help command
   
   `help, data`

7. Type the following on the ENVI command line to a BIP slice of data for the last line.
   
   ```
   data = envi_get_slice(fid=fid, pos=pos, line=dims[4], 
   xs=dims[1], xe=dims[2], /bip)
   ```

   This interactive example is for demonstration purposes only. In practice, these steps are part of the processing routine.

### Processing Status Report

The processing status dialog shows the percent completed for the current processing. Using the ENVI processing status reports, the developer has control over the increment size and the update frequency. The optional “Cancel” button is used to abort processing on the next increment update. The processing status is controlled by three procedures which initialize, set the increment, and update the status. These routines are listed in Table 4-5.

**Note**

Processing is aborted on the next increment update unless the increment is 100% or the final increment is processing, then “Cancel” is ignored.

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_REPORT_INC</td>
<td>Set the report increment</td>
<td>488</td>
</tr>
<tr>
<td>ENVI_REPORT_INIT</td>
<td>Initialize the report dialog</td>
<td>489</td>
</tr>
<tr>
<td>ENVI_REPORT_STAT</td>
<td>Update the percent completed and check for cancel</td>
<td>491</td>
</tr>
</tbody>
</table>

*Table 4-5: Processing status routines available in ENVI.*

Each element of the string array argument to ENVI_REPORT_INIT is displayed on a separate line in the status dialog. ENVI typically uses a two element array indicating the input and output file. To remain consistent, the array would be set as follows, where *filename* is the actual input or output file name:

```
['Input File: filename', 'Output File: filename']
```
For items that are output to memory, the array is set as follows.

['Input File: filename', 'Output to Memory']

The status dialog is usually updated inside the tiling loop, which loops over the total number of tiles. For these cases the report increment is set to the total number of tiles. For example, for five tiles, the report increment would be set to five, which implies 20% increments.

**Example: Processing Status Dialog**

This example interactively creates a processing status dialog and updates the percent completed.

1. Start ENVI.
2. Open the IDL development environment (PC, Macintosh) or the shell window in which ENVI was started (UNIX).
3. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to create the Status dialog.

   ```plaintext
   ```

   The following dialog should now be on the screen.

   ![Figure 4-21: Processing status dialog.](image)

   Now set the increment to three and then update the percent completed using ENVI_REPORT_STAT.

4. Type the following on the ENVI command line to set the increment.

   ```plaintext
   envi_report_inc, base, 3
   ```
5. Type the following on the ENVI command line to update the percent completed to 33%, 66% and 99%.

   envi_report_stat, base, 1, 3
   envi_report_stat, base, 2, 3
   envi_report_stat, base, 3, 3

6. Type the following on the ENVI command line to delete the status dialog.

   envi_report_init, base=base, /finish

This interactive example is for demonstration purposes only. In practice, these steps are part of the processing routine. When used with a tiled processing routine, the increment is typically the number of tiles being processed and the report is updated before the next tile is requested. For non-tiled processing, the increment and report updates can be related to any processing loop used.
Adapting User Functions for ENVI RT

User functions can still be added to Runtime versions of ENVI, even though access to IDL is prohibited. Because ENVI Runtime can not compile ASCII .pro files, the compiled user function routines must be stored in binary .sav files. Before creating the save file, the user function code may need some important modifications.

**Using FORWARD_FUNCTION or COMPIL"OPT STRICTARR**

Because the ENVI library functions are unknown to the IDL compiler, any references to them will erroneously be interpreted as variable names. If you have written your IDL code using square brackets for variable dereferencing (the recommended syntax beginning with IDL 5.0), this problem can be eliminated by using the COMPIL"OPT statement with the STRICTARR argument. If you used parentheses to dereference variables (as was required before IDL 5.0), you must use a FORWARD_FUNCTION statement to declare the names of the ENVI functions so that the compiler doesn’t treat them as variables.

**Using RESOLVE_ALL to Find and Compile Dependent Routines**

Many common IDL procedures aren't actually built into IDL (as part of the binary IDL application), but are included in the IDL library (or *lib*) as .pro code. For example, XMANAGER, CONGRID, and SWAP_ENDIAN are all IDL *lib* routines. When working in ordinary IDL or ENVI, you usually don't notice if a routine is built-in or part of the *lib*, because IDL automatically compiles needed routines whenever necessary. However, because ENVI RT can't compile any .pro code, if your user function includes any *lib* routines they must be compiled in the IDL session before making the save file. Because this situation occurs quite often, IDL provides a special routine called RESOLVE_ALL to find and compile all dependent routines in any compiled procedure.

Using RESOLVE_ALL on ENVI user functions is a bit different than using it on ordinary IDL procedures. RESOLVE_ALL will only find and compile dependent routines that are stored in .pro files, although the ENVI library routines are stored in binary .sav files. Thus, when RESOLVE_ALL finds a reference to one of the ENVI library routines it will not be able to compile, causing the routine to halt and issue an error. Of course, the ENVI library routines do not need to be included in the user function's save file because they will be compiled and available when ENVI is running. In order to be able to use RESOLVE_ALL on a user function that includes ENVI library routines, you must call it with the CONTINUE_ON_ERROR keyword.
set, which allows it to run through the entire user function without halting when an error occurs. Each ENVI routine that it was unable to compile will be listed in an error message in the IDL Command Log.

Making a Save File

1. Immediately following the procedure definition statement in the user function code, add the COMPileeOPT STRICTARR or FORWARD_FUNCTION statement.

   • If using the FORWARD_FUNCTION statement, name all of the ENVI library functions used in the code (ENVI procedures do not need to be listed, only those that are functions).

2. Remember to save the modified user function code.

3. Start a new IDL session.

4. Compile the modified user function procedure.

5. Call RESOLVE_ALL to compile all dependent procedures in the code:

   IDL> Resolve_All, /continue_on_error

   If your user function uses any ENVI library routines, expect to see several error messages printed to the Command Log.

6. Make the save file by calling the SAVE procedure with the ROUTINES keyword set:

   IDL> SAVE, file='my_user_function.sav', /routines

   The name of the save file that is created must have the same root name as the name of the user function procedure, must have a .sav extension, and must be placed in ENVI Runtime’s SAVE_ADD directory.
Chapter 5: Programming Tools

This chapter covers the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plotting</td>
<td>125</td>
</tr>
<tr>
<td>Reports</td>
<td>127</td>
</tr>
<tr>
<td>RGB Colors Triplets</td>
<td>128</td>
</tr>
<tr>
<td>File Information</td>
<td>129</td>
</tr>
<tr>
<td>Managing Files</td>
<td>131</td>
</tr>
</tbody>
</table>
This chapter covers programming tools used in custom development including plotting, reports, graphics colors, and general file utilities. Plots and reports can display or summarize processing results while the other tools are provided as utilities to the developer. Detailed descriptions and examples are found in the following sections.
Plotting

ENVI provides the user with the access to the ENVI plot window. The user can define a set of plots and load them into a plot window with a single call. ENVI manages the resulting plot providing the functionality common to any ENVI plot window. The loaded data can be spatial (like an X-profile), spectral (such as a Z-profile), or any X,Y data.

The routine ENVI_PLOT_DATA (see “ENVI_PLOT_DATA” on page 481) is used to plot X,Y data. Any number of Y plots can be specified but they all use the same X values. Optional keywords are used to set the plot title, color, name, line style, axes names and titles and provide custom control for the displayed data. The following example details the use of ENVI_PLOT_DATA.

Figure 5-1: Example ENVI plot.
Example: Plotting Data

This example interactively creates an X,Y plot using a single Y data array. The function FINDGEN is used to create an X array of incrementing floating point numbers. The Y values are an array of uniform random numbers calculated using the function RANDOMU. The X,Y data is plotted using ENVI_PLOT_DATA with a number of optional keywords.

1. Start ENVI.
2. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).
3. Type the following on the ENVI command line to create the Status Dialog (the line that contains the prompt “ENVI>”).

```idl
x = findgen(100)
y = randomu(seed, 100)
envi_plot_data, x, y, plot_title='Numbers', $ 
   plot_colors=[10], plot_names='Random', $ 
   xtitle='Count', ytitle='Value'
```

The following plot is displayed on the screen. Note that the value of the PLOT_COLORS keyword references a ENVI graphic colors index (see “RGB Colors Triplets” on page 128).

This interactive example is for demonstration purposes only. In practice, these steps are part of the processing routine; the actual X,Y data values vary with the application.
Reports

ENVI provides the user with access to the ENVI report widget which is used to display text data. The report widget generated by ENVI_INFO_WID displays each element of a user defined string array on a new line. Once displayed, ENVI automatically manages the resulting report, providing the functionality common to all report widgets, including output to a file. The following example details the use of ENVI_INFO_WID.

Example: Creating a Report

This example interactively creates a text report and displays the result. A string array with four elements, including one NULL string, is created for output to the report widget. The NULL string element translates into a blank line.

1. Start ENVI.
2. Open the IDL development environment (PC, Macintosh) or the shell window in which ENVI was started (UNIX).
3. Type the following at the ENVI command line (the line that contains the prompt “ENVI>”) to create the Report window.

   ```idl
   str = ['Line 1', 'Next line is blank', '', 'Line 4']
   envi_info_wid, str, title='Report'
   ```

The following report is displayed on the screen.

This interactive example is for demonstration purposes only. In practice these steps are part of the processing routine.

![Example ENVI report](image)

Figure 5-2: Example ENVI report.
RGB Colors Triplets

ENVI maintains a set of graphics colors that are used for annotations, vector overlays, plots, classification images, and other items. Many routines reference colors by the graphics color index, however, some use the RGB color triplet. The routine ENVI_GET_RGB_TRIPLETS returns the RGB value for any color index. To avoid indexing past the number of graphics colors the modulo operator is automatically applied within ENVI_GET_RGB_TRIPLETS. For example, to set the lookup table for a classification image, ENVI_GET_RGB_TRIPLETS is called for each class with the index set equal to the class number. If there are more classes then color indexes the class colors are repeated as necessary.

Note

Users can add custom graphics colors to ENVI by modifying the colors.txt file (see Editing System Color Tables in the ENVI User’s Guide). Each line of the file contains an RGB triplet value and a name. Although not required, it is best to leave graphics color 0 as black and 1 as white.

Example: Getting RGB Color Values

This example interactively gets RGB color triplets associated with a graphics color and prints the result to the IDL log window.

1. Start ENVI.
2. Open the IDL development environment (PC, Macintosh) or the shell window in which ENVI was started (UNIX).
3. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to get the RGB for color index zero.
   ```idl
   envi_get_rgb_triplets, 0, r, g, b
   print, r, g, b
   ```
4. Type the following at the ENVI command line to get the RGB for color index one.
   ```idl
   envi_get_rgb_triplets, 1, r, g, b
   print, r, g, b
   ```

This interactive example is for demonstration purposes only. In practice these steps are part of the processing routine.
File Information

Extracting file information is an integral part of ENVI programming. The routine ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411) is designed to provide all known file information for any open file, whether in memory or on disk. Information ranges from the dimensions of the data to map coordinates, and includes a mechanism for extracting custom information. File information is set with the routine ENVI_SETUP_HEAD (see “ENVI_SETUP_HEAD” on page 510) or ENVI_ENTER_DATA (see “ENVI_ENTER_DATA” on page 389).

Some of the returned values from ENVI_FILE_QUERY (like H_INFO) are handles to the actual data. Access to the data is achieved using HANDLE_VALUE (see “HANDLE_VALUE” on page 560).

Example: Basic Image Information

This example interactively chooses a file, extracts basic file information using ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411) and prints the result to the IDL log window.

1. Start ENVI.
2. Open any image file.
3. Open the IDL development environment (PC, Macintosh) or the shell window in which ENVI was started in (UNIX).
4. Type the following at the ENVI command line (the line that contains the prompt “ENVI>”) to select an input file.

   ```idl```
   envi_select, title='Input Filename', fid=fid
   ```idl```

5. Type the following at the ENVI command line to get the basic file information and print the result.

   ```idl```
   envi_file_query, fid, ns=ns, nl=nl, nb=nb, offset=offset,$
   data_type=data_type, interleave=interleave
   print, ns, nl, nb, offset, data_type, interleave
   ```idl```

This interactive example is for demonstration purposes only. In practice these steps are part of the processing routine.
Example: Map Information

This example interactively chooses a file, extracts the map projection information, and prints the result to the IDL log window. A georeferenced file must be chosen using ENVI_SELECT (see “ENVI_SELECT” on page 500); otherwise there is no associated projection information. The routine ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411) returns the handle to the associated map projection structure and the function HANDLE_VALUE returns the structure variable.

Note

The function HANDLE_INFO is used to check for valid handles. This function returns 0 for an invalid handle and 1 for a valid handle.

1. Start ENVI.
2. Open the IDL development environment (PC, Macintosh) or the shell window in which ENVI was started (UNIX).
3. Open an image file that is georeferenced.
4. Type the following at the ENVI command line (the line that contains the prompt “ENVI>”) to select an input file.

   envi_select, title='Georeferenced Filename', fid=fid

5. Type the following at the ENVI command line to get the basic file information.

   envi_file_query, fid, ns=ns, nl=nl, nb=nb, h_map=h_map

6. Type the following at the ENVI command line to get the map information and print the result.

   handle_value, h_map, map_info
   print, map_info

7. Type the following at the ENVI command line to get projection information and print the result.

   proj = envi_get_projection(h_map=h_map)
   print, proj

This interactive example is for demonstration purposes only. In practice these steps are part of the processing routine.
Managing Files

ENVI provides the tools to open, close, and select image files both for interactive and batch mode programming. Interactive programs select ENVI image files using ENVI_SELECT and the returned FID is used as a reference to the file. Image files used for display and processing must be referenced by their FID. When selecting non-image files, like ROI files, use ENVI_PICKFILE to get the filename. In the non-image case, the filename provides the link to extract the necessary information. For example, the routine ENVI_FILE_QUERY references the file with an FID, while the routine ENVI_RESTORE_ROIS uses a filename to restore a set of ROIs.

When performing batch mode programming, the routine ENVI_OPEN_FILE opens any ENVI image file and returns the FID without any user interaction. Both batch and interactive programming can use ENVI_FILE_MNG to close and optionally delete image files. For examples on managing files in batch see “Examples of ENVI Batch Programming” on page 65.

The following table lists the routines used to manage ENVI files.

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_FILE_MNG</td>
<td>Manage open files</td>
<td>410</td>
</tr>
<tr>
<td>ENVI_OPEN_DATA_FILE</td>
<td>Open a external format image file</td>
<td>473</td>
</tr>
<tr>
<td>ENVI_OPEN_FILE</td>
<td>Open an image file</td>
<td>476</td>
</tr>
<tr>
<td>ENVI_OUTPUT_TO_EXTERNAL_FORMAT</td>
<td>Output an ENVI file to an external format</td>
<td>477</td>
</tr>
<tr>
<td>ENVI_PICKFILE</td>
<td>ENVI file selection</td>
<td>479</td>
</tr>
<tr>
<td>ENVI_SELECT</td>
<td>Select an open ENVI image file</td>
<td>500</td>
</tr>
</tbody>
</table>

*Table 5-1: Routines for managing files in ENVI.*
Example: Choosing Files Interactively

This example interactively chooses a file using ENVI_SELECT (see “ENVI_SELECT” on page 500) and then closes the file using ENVI_FILE_MNG (see “ENVI_FILE_MNG” on page 410). With ENVI started run ENVI_SELECT from the command line. Once the file selection dialog appears, either select any open file or select “Open Image File” if no files are currently open. Once an image file is selected press “OK”. The returned FID is then used to close the file with the routine ENVI_FILE_MNG.

If the “Cancel” button is selected on the ENVI_SELECT dialog, a -1 is returned in the FID keyword indicating no file was selected.

1. Start ENVI.
2. Open the IDL development environment (PC, Macintosh) or the shell window in which ENVI was started (UNIX).
3. Type the following at the ENVI command line (the line that contains the prompt “ENVI>”) to select an input file.
   ```idl```
   ```
   envi_select, title='Input Filename', fid=fid
   ```
   ```idl```
4. Type the following at the ENVI command line to close the file.
   ```idl```
   ```
   envi_file_mng, id=fid, /remove
   ```
   ```idl```

This interactive example is for demonstration purposes only. In practice these steps are part of the processing routine.
Chapter 6:
Interactive User Routines

This chapter covers the following topics:

- Plot Function ....................... 135
- Spectral Analyst Functions ............ 138
- User Defined Map Projection Types ..... 142
- User Defined Units .................. 147
- User Move Routines ................. 148
This chapter covers the development of routines that hook into ENVI’s interactive analysis. For selected functions, ENVI allows users to develop additional methods or transforms that are applied automatically to the data. Interactive routines are triggered by certain events or user selection. For example, when selected, a custom plot function applies a transformation to the plot data and displays the result in place of the original data. There are also ways to apply custom scoring routines to the Spectral Analyst™ and attach a user procedure to the zoom event. Detailed descriptions and examples are found in the following sections.
Plot Function

Plot Functions provide a method for applying transforms or functions to data in any plot window. When a plot function is selected, the normal data is input into the function and the resulting data is displayed in the plot window (see Interactive Plot Functions in the Users Guide). New data also has the function applied prior to its display. Usually plot functions are applied to spectral data from Z-profiles, spectral libraries, ROI means, or other sources, however, there are no data source requirements to these functions.

ENVI provides a standard set of plot functions and also allows users to define custom plot functions. Custom plot functions receive the normal plot data as input, apply the processing, and return the transformed data to the plot. Any new data is also applied through the selected plot function. Plot functions only operate on the Y axis data; the X axis remains the same.

Custom plot functions are added to ENVI by entering the menu name and function into the useradd.txt file in the menu directory of the installation. Plot functions use the {plot} tag to differentiate them from other functions. The format for a plot function is:

```plaintext
{plot} {Button Name} {function_name} {type=n}
```

where

- `{plot}` - Tag to indicate the following definition is a plot function.
- `{Button Name}` - Menu button name for the Plot Functions pulldown menu.
- `{function_name}` - Name of the plot function to call.
- `{type=n}` - Type of plot function updates. Set type=0 to call the plot function only when new data is available. Set type=1 to call the plot function when new data is available or the plot is zoomed.

A sample portion of the useradd.txt file related to plot functions is shown below.

```plaintext
{plot} {Normal} {sp_normal} {type=0}
{plot} {Continuum Removed} {sp_continuum_removed} {type=1}
{plot} {Binary Encoding} {sp_binary_encoding} {type=0}
```

To add a custom plot function that only updates with new data the following line would be added to the file.

```plaintext
{plot} {My function} {my_plot_func} {type=0}
```
To change the previous custom plot function to update with new data or when the data is zoomed, change the type to one as shown by the following definition.

\{plot\} \{My function\} \{my_plot_func\} \{type=1\}

Plot functions declarations have a number of arguments including the X and Y data, bad band information, and the left and right zoom indexes. Remember, ENVI calls your plot function and automatically passes the required parameters into it. A sample plot function declaration is shown below.

```
FUNCTION my_func, X, Y, BBL, BBL_ARRAY, L_POS=L_POS, $
               R_POS=R_POS, _EXTRA=_EXTRA
```

where

- my_func - The plot function name.
- X - Data values for the X axis.
- Y - Data values for the Y axis.
- BBL - Pointer to each of the bad bands in a Z-profile. If there are no bad bands or the current plot is not a Z-profile this value is undefined.
- BBL ARRAY - An array of ones and zeros representing the good and bad points in the plot respectively. The number of elements of BBL ARRAY equals the number of elements of X. This array is defined for all plots regardless of the type of data.
- L_POS - The left (lower) index of the X array.
- R_POS - The right (upper) index of the X array.
- EXTRA - You must specify this keyword to collect all extra keywords that ENVI uses when calling the user-defined function.

The new Y data is returned from the plot function. The following model is used for plot functions.

```
FUNCTION my_plot_func, X, Y, BBL, BBL_ARRAY, $
               L_POS=L_POS, R_POS=R_POS, _EXTRA=_EXTRA
               statements ...
               RETURN, plot_result
END
```

Plot functions that operate on the Z-profile data may be concerned with the bad bands information passed to the routine. Bad bands are typically ignored in calculation of the plot function and their output values are not displayed in the plot.
To ignore bad bands, define a pointer to the good bands and then use this as an index into the X and Y arrays. The following statement sets the variable PTR to an index of all good points and the variable COUNT equals the number of good points.

```plaintext
ptr = where(bbl_array eq 1, count)
```

The following example further illustrates the implementation of a plot function.

### Example: Plot Function

This example creates a plot function to compute the zero mean value of a plot (i.e., center the plot). The mean value is calculated only on the good points in the plot and does not change when the plot is zoomed. First the good data points are found by examining the BBL_ARRAY and building an index where it is equal to one. Next only the good points are used to calculate the mean which is subtracted from the Y axis data. If no valid data points exist then the result is set to zero. The resulting Y data is returned from the function.

The sample code listing follows and can be found in the file `irplot.pro` in the `lib` directory of the installation.

```plaintext
function pf_zero_mean, x, y, bbl, bbl_array, _extra=_extra
  bbl_ptr = where(bbl_array eq 1, count)
  if (count gt 0) then $
    result = y - (total(y[bbl_ptr]) / count) $
  else $
    result = fltarr(n_elements(y))$
  return, result
end
```

Save the routine to a file and place in the `save_add` directory of the installation tree. In addition, the following item must be added to the `useradd.txt` file in the `menu` directory of the installation tree. This will allow selection of the function from the *Plot Functions* pulldown menu.

```plaintext
{plot} {Zero Mean} {pf_zero_mean} {type=0}
```

The following steps outline the procedure for executing this example:

1. Save the routine to a file and place in the `save_add` directory.
2. Add the plot function definition to the `useradd.txt` file.
3. Start ENVI.
4. Open a file and display a X-profile.
5. Select *Zero Mean* from the *Plot Functions* pulldown menu in the X-profile plot.
Spectral Analyst Functions

Spectral Analyst™ functions provide a method to match an unknown spectrum to the materials in a spectral library. ENVI includes common spectral similarity techniques like binary encoding, spectral angle mapper, and spectral feature fitting. Custom functions can be added to the Spectral Analyst and used alongside those defined in ENVI. Individual 0 to 1 scores are accumulated for each of the functions and the library spectra are ranked in order of best to worst match.

Custom spectral analyst functions are added to ENVI by entering the menu name and function into the useradd.txt file in the menu directory of the installation. Spectral analyst functions use the {identify} tag to differentiate them from other functions in this file. The format for an entry is:

{(identify) (Method Name) (Out Name) (func_name) (min, max)}

where

{(identify)} - Tag to indicate the following definition is a spectral analyst function.

{(Method Name)} - Method name for the spectral identification widget.

{(Out Name)} - Column name for the spectral analyst output ranking report window.

{(func_name)} - Name of the spectral identification function to call. The function name is also used as the base name for the identification setup function, function_name_setup, called once for a given spectral library.

{(min, max)} - The default minimum and maximum outputs for the identification function. The default minimum and maximum values may be edited when running the spectral analyst. The current scale factors are passed into the identification function. The output values are then scaled to a [0,1] range to allow a cumulative ranking of all methods.

Spectral analyst functions have two parts. The first is the setup procedure which is called after the spectral library is selected and conditions the library for the selected identification routine. All library calculations that are not dependent on the input data should be performed once in the setup procedure. The name of the setup procedure is formed from the function name with an additional “_SETUP”. For example, the setup procedure for FUNC_NAME is FUNC_NAME_SETUP.
Note

Unlike the identification function, the setup routine is a procedure and must be defined even if no setup is necessary.

The setup procedure for the function FUNC_NAME is declared as follows:

```en
PRO func_name_setup, WL, SPEC_LIB, HANDLES, NUM_SPEC=NUM_SPEC
```

where

- **WL** - The wavelength values for the spectral library. All spectra in the library have one sample at each wavelength, allowing a single wavelength vector for each library.

- **SPEC_LIB** - A two dimensional array of all spectral library spectra. The dimension are [wavelength_samples, num_spectra].

- **HANDLES** - An array of two handles for storage of user data. Any preprocessing of the spectral library should be stored in one of these handles. The handle array is also passed into the identification function.

- **NUM_SPEC** - The number of spectra in the library. This value is equal to the second dimension of the SPEC_LIB array.

The second part of the spectral analyst function is the identification function which calculates the score for the current spectrum against the library spectra. The method for calculating the score depends on the user implemented function. The current minimum and maximum scale factors must be applied to the resulting score. The function output is an array of scaled scores for each of the library spectra. The identification function for FUNC_NAME is declared as follows:

```en
FUNCTION func_name, WL, REF_SPEC, SPEC_LIB, HANDLES,$
NUM_SPEC=NUM_SPEC, SCALE_VALS=SCALE_VALS
```

where

- **WL** - The wavelength values for the spectral library. All spectra in the library have one sample at each wavelength, allowing a single wavelength vector for each library.

- **REF_SPEC** - The reference spectrum used to score against the library. The reference spectrum has the same number of wavelength samples as the library spectra.

- **SPEC_LIB** - A two dimensional array of all spectral library spectra. The dimension are [wavelength_samples, num_spectra].
HANDLES - An array of two handles for storage of user data. Any data stored in the setup procedure can be extracted using: HANDLE_VALUE.

NUM_SPEC - The number of spectra in the library. This value is equal to the second dimension of the SPEC_LIB array.

SCALE_VALS - An array containing the current minimum and maximum scale factors, respectively. The scale factors are applied to the calculated score to bring its range to between zero and one.

The combination of the setup procedure and the identification function forms a spectral analyst function. The resulting score is automatically combined with other spectral analyst functions based on the current weights. Any spectral analyst function with a weight of zero is not called since its output score would not be used.

The following example further illustrates the implementation of a spectral analyst function.

Example: Spectral Analyst Function

This example creates a spectral analyst function to calculate the minimum distance between each library spectra and the current reference spectrum. The setup procedure is defined but empty since no setup is necessary in this example. The identification function computes the distance measure as the square root of the sum of the differences at each wavelength. A distance score is computed between each library spectrum and the input reference spectrum. The output scores are scaled by the largest distance error in order to keep the distance measure between zero and one.

Note

This example is for illustration purposes only. An actual minimum distance spectral analyst function would remove the continuum from the library (in the setup procedure) and reference spectrum prior to calculating the distance score.

The sample code listing follows and can be found in the file irsadist.pro in the lib directory of the installation.

```pro
pro irdist_func_setup, wl, spec_lib, handles, num_spec=num_spec ; No initialization is necessary end

function irdist_func, wl, ref, spec_lib, handles, num_spec=num_spec,$
  scale_vals=scale_vals
  ; Compute the distance compared to each library member
  result = dblarr(num_spec)
```
for i=0L, num_spec-1 do $
  result[i] = sqrt(total((spec_lib[*',i]-ref)^2, /double))$

; scale the result from zero to one
dmax = max(result, min=dmin)
return, (1d - ((result - dmin) / (dmax - dmin))) / scale_vals(1)
ext

Save the routine to a file and place in the save_add directory of the installation tree. In addition, the following item must be added to the useradd.txt file in the menu directory of the installation tree to allow selection of the function in the Spectral Analyst.

{identify} {Minimum Distance} {MDIST} {irsadist_func} {0,1.}

The following steps outline the procedure for executing this example.

1. Save the routine to a file and place in the save_add directory.
2. Add the Spectral Analyst function definition to the useradd.txt file.
3. Start ENVI.
4. Open a file and display a Z-profile.
5. Start the Spectral Analyst under the Spectral Tools pulldown menu and open a spectral library for comparison.
6. Set the Minimum Distance weight to 1.0 and select “OK”.
7. Rank the current Z-profile by selecting “Apply”.
User Defined Map Projection Types

ENVI supports many different map projections and map projection types. You can create customized map projections using the ENVI function Utilities > Map Projection Utilities > Build Customized Map Projection from the supported map projection types. However, you may want to define your own map projection type. In this case, user-defined projection types can be added to ENVI by writing an IDL procedure that calculates the forward and inverse conversions between latitude/longitude and the new projection coordinates.

Note
For information about ENVI’s datum, ellipsoids, and map projections, see the online help or the ENVI User’s Guide.

To define your new projection:

1. Select Utilities > Map Projection Utilities > Build Customized Map Projection and select your new projection type name from the “Projection Types” list.

2. When the dialog appears, enter your input parameters.

Your projection will be added to the “map_proj.txt” file when you run the Build Customized Map Projection function.

Add custom map projection types to ENVI by entering the projection name and routine name in the useradd.txt file in the menu directory of the installation. Map projection types use the \{projection type\} tag to differentiate them from other functions in this file. The format for an entry is:

\{projection type\} \{projection name\} \{routine_root_name\} \{number of extra parameters\}

where

\{projection type\} - Tag to indicate the following definition is a user defined projection type.

\{projection name\} - Name of the projection in the “Projection Type” list in ENVI.

\{routine_root_name\} - Root name for the user procedures.

\{number of extra parameters\} - Number of parameters this projection will require in addition to the default parameters which are: ellipsoid a and b; projection origin (lat/lon); and false easting/northing. This value can be \{0\} if there...
User defined map projection routines may have two parts, one which allows you to enter extra parameters, and the other that does the coordinate conversions. If extra parameters are needed to define the projection, then a procedure named “routine_root_name”_DEFINE is used to input these parameters. This procedure contains one parameter which is a DBLARR of the number of extra parameters needed. The extra parameters are double-precision floating point data values and up to 9 can be input. The define procedure assigns the values to the extra parameters by either getting input from a dialog when the Build Customized Map Projection function is run or by setting the values in the routine.

Note
If the number of extra parameters is {0}, then the “routine_root_name” procedure is not needed.

The second procedure needed is named “routine_root_name”_CONVERT. It performs the forward and inverse lat/lon to new projection coordinate conversions. This routine must work for both a scalar value or an array of values. This routine has 6 parameters, as shown below.

```
routine_root_name_convert, x, y, lat, lon, TO_MAP=TO_MAP,
PROJECTION=PROJ
```

where

- x, y - are the map projection coordinates
- lat/lon - are the latitude and longitude coordinates
- TO_MAP - if set, the routine will convert lat/lon to map x, y; if not set, the routine will convert from map x, y to lat/lon
- PROJ - the (environ_struct) of the user defined projection where

```
proj.type=99
proj.params[0]=a - ellipsoid a value
proj.params[1]=b - ellipsoid b value
proj.params[2]=lat0 - projection origin latitude
proj.params[3]=lon0 - projection origin longitude
proj.params[4]=x0 - false easting
proj.params[5]=y0 - false northing
proj.params[6:*] - additional user parameters
```

The .pro or .sav file for the routines need to be added to the save_add directory of the ENVI installation.
To complete the projection definition process:

1. Restart ENVI.
2. Select Utilities > Map Projection Utilities > Build Customized Map Projection.
3. The newly defined projection type will appear in the list of Projection Types.
4. Select the projection type and enter the values for the parameters.

The function saves the new projection to the map_proj.txt file and it can be selected from all the ENVI projection functions.

**Example: User Defined Map Projection**

The following two examples of user defined map projections illustrate how to define a new projection without any additional parameters, USER_PROJ_TEST1, and with four additional parameters, USER_PROJ_TEST2. Depending on the type of projection being added use one of the two examples as a model.

For the sake of illustration the first test passes through the map coordinates by setting the output map coordinates equal to the input coordinates. Since there are no additional parameters only the _convert routine is needed. The example code for TEST1 follows:

```plaintext
pro user_proj_test1_convert, x, y, lat, lon, to_map=to_map, projection=p
if (keyword_set(to_map)) then begin
  x = lon
  y = lat
end if else begin
  lon = x
  lat = y
end else
end
```

Save the routine to a file and place in the save_add directory of the installation tree. In addition, add the following to the useradd.txt file in the menu directory of the installation tree to allow the new user defined projection.

```plaintext
{projection type} {User Projection #1} {user_proj_test1} {0}
```

The following steps outline the procedure for executing this example:

1. Save the routine to a file and place in the save_add directory.
2. Add the User Defined Projection definition to the useradd.txt file.
3. Start ENVI.
4. Define your new projection *Utilities > Map Projection Utilities > Build Customized Map Projection.*

5. Select your new projection type name from the “Projection Types” list.

6. After successfully entering any parameters you can save your new projection to map_proj.txt

The next example requires the definition of four additional parameters. Since there are additional parameters you must create both the _define and _convert routines.

The example code for TEST1 follows:

```plaintext
pro user_proj_test2 Define, add_params
  if (n_elements(add_params) gt 0) then begin
    default_1 = add_params[0]
    default_2 = add_params[1]
    default_3 = add_params[2]
    default_4 = add_params[3]
  endif
  base = widget_auto_base(title='User Projection #1 Additional Parameters')
  sb = widget_base(base, /column, /frame)
  sb1 = widget_base(sb, /row)
  wp = widget_param(sb1, prompt='Parameter #1', xsize=12, dt=4,$
    field=4, $
    default=default_1, uvalue='param_1', /auto)
  sb1 = widget_base(sb, /row)
  wp = widget_param(sb1, prompt='Parameter #2', xsize=12, dt=4,$
    field=4, $
    default=default_2, uvalue='param_2', /auto)
  sb1 = widget_base(sb, /row)
  wp = widget_param(sb1, prompt='Parameter #3', xsize=12, dt=4,$
    field=4, $
    default=default_3, uvalue='param_3', /auto)
  sb1 = widget_base(sb, /row)
  wp = widget_param(sb1, prompt='Parameter #4', xsize=12, dt=4,$
    field=4, $
    default=default_4, uvalue='param_4', /auto)
  result = auto_wid_mng(base)
  if (result.accept) then $
    add_params = [result.param_1, result.param_2, $
      result.param_3, result.param_4]
end
```
Chapter 6: Interactive User Routines

User Defined Map Projection Types

pro user_proj_test2_convert, x, y, lat, lon, to_map=to_map, projection=p
if (keyword_set(to_map)) then begin
   x = lon * 100. + p.params[4]
   y = lat * 100. + p.params[5]
endif else begin
   lon = (x - p.params[4]) / 100.
   lat = (y - p.params[5]) / 100.
endelse
end

Save the routine to a file and place in the save_add directory of the installation tree. In addition, the following item must be added to the useradd.txt file in the menu directory of the installation tree to allow the new user defined projection.

(projection type) {User Projection #2} {user_proj_test2} {4}

The following steps outline the procedure for executing this example.

1. Save the routine to a file and place in the save_add directory.
2. Add the User Defined Projection definition to the useradd.txt file.
3. Start ENVI.
5. Select your new projection type name from the “Projection Types” list.
6. After successfully entering any parameters you can save your new projection to map_proj.txt
User Defined Units

ENVI has different units that can be selected when using map projections or using ENVI’s measurement tools. These units include meters, kilometers, feet, yards, miles, nautical miles, acres, hectares, degrees, minutes, seconds, and radians. ENVI allows you to define your own units for use in map projections or for measurement calculations. These units are defined by entering a scale factor that converts between the user units and meters, degrees, or meters^2.

Custom units are added to ENVI by entering a scale factor into the useradd.txt file in the menu directory of the installation. Unit definitions use the {units} tag to differentiate them from other functions in this file. The format for an entry is:

```
{units} {Name} {scale factor} {0|1|2}
```

where

- `{units}` - Tag to indicate the following definition is a user defined unit.
- `{Name}` - unit name
- `{scale factor}` - multiplication scale factor to convert the new units to meters, degrees, or meters^2
- `{0|1|2}` - flag value telling which units the scale factor converts to, where 0=meters, 1=degrees, 2=meters^2

Examples of each type of unit definition are shown below. Each of these lines would be added to the “useradd.txt” file:

```
{units} {Feet} {0.3048} {0} ; distance with respect to meters
{units} {Minutes} {0.016666667} {1} ; angular distance with respect to degrees
{units} {Acres} {4046.873} {2} ; area with respect to meters^2
```
User Move Routines

Two types of user move routines provide methods of attaching user functions to motion events in the display window or to zoom positioning events. ENVI will call a User Defined Motion Routine each time the cursor moves in the display window and a User Define Move Routine will be called each time the zoom location is moved. For both move routines the positional information is passed into the routine allowing the display of positional dependent information. For example, move routines could display housekeeping data for the current line of an image.

Most move routines display the user data in a text widget, much like the cursor location/value widget. The move routine first checks if the widget exists, and if not, creates the widget. Keep in mind that the widget may have been closed since the last update. Displayed data can come from the current image or another source and it’s possible to prompt for an input file when move routine widget is created.

Move routines are defined in the ENVI configuration file, `envi.cfg`, or by selecting `File > Preferences` from the ENVI main menu. Once a move routine is defined it will be called for every zoom location event regardless of the image being displayed. Custom move routines may also use the file type in order to further restrict the display of data. When displaying housekeeping data from a custom format, a new file type should be added to the `filetype.txt` file in the `menu` directory of the installation tree. The move routine can then use `ENVI_FILE_TYPE` (see “ENVI_FILE_TYPE” on page 417) to check for the proper file type prior to displaying data.

User move routines can also output data directly to the IDL log window using a simple PRINT statement. Although not elegant, it’s a simple way to develop or debug move routines that will eventually use a widget interface.

Move routines are defined as procedures with parameters for the display number and the X and Y locations. The XY locations are floating point values and may be fractional portions of a pixel. Also, the keywords XSTART and YSTART define the X and Y starting location of the first pixel in the file. The Motion routine is identical with the addition of an EVENT keyword which contains the event structure of the drawable so that you can use event.x, event.y, event.press, etc.

```
PRO user_move, DN, XLOC, YLOC, XSTART=XSTART, YSTART=YSTART
where

    DN - The display number where the zoom event occurred within.

    XLOC - The current X location in image coordinates. The XSTART value must be subtracted to convert to file coordinates.
```

User Move Routines

ENVI Programmer’s Guide
YLOC - The current Y location in image coordinates. The YSTART value must be subtracted to convert to file coordinates.

XSTART - The X starting location of the first pixel in the file (in image coordinates).

YSTART - The Y starting location of the first pixel in the file (in image coordinates).

The X and Y locations are in image coordinates and the pixel starting locations must be subtracted to convert to file coordinates. The file coordinates are necessary for using the function ENVI_GET_DATA, but may not be needed for extracting custom housekeeping data.

**Note**

When bands from different files are displayed as an RGB the X and Y starting location should be retrieved from ENVI_FILE_QUERY instead of using the keyword inputs.

The following examples further illustrate the implementation of a user define move routine.

**Example: Simple User Defined Move Routine**

This example create a user defined move routine that outputs the current zoom position and pixel value to the IDL log window. Using the DN the routine ENVI_DISP_QUERY (see “ENVI_DISP_QUERY” on page 380) returns the FIDs, displayed band for the current display. The COLOR keyword to ENVI_DISP_QUERY is used to determine if a grayscale or RGB is currently displayed. For each displayed band a DIMS array is created and used as input into ENVI_GET_DATA (“ENVI_GET_DATA” on page 423) to retrieve the value of the current pixel. The pixel value is printed to the “IDL log window.

The sample code listing follows and can be found in the file irudmv1.pro in the lib directory of the installation.
pro ud_move_1, dn, xloc, yloc, xstart=xstart, ystart=ystart
; Get the file FIDs
  envi_disp_query, dn, fwd=fid, pos=pos, color=color
  if (color eq 8) then nb = 3 $ 
else nb = 1
; Print the DN and zoom location
print, dn, xloc, yloc
; Print out the current pixel for each displayed band
for i=0, nb-1 do begin
  envi_file_query, fid[i], xstart=xstart, ystart=ystart dims =
  long([0, $
    xloc - xstart - 1, xloc - xstart - 1, $
    yloc - ystart - 1, yloc - ystart - 1])
  print, envi_get_data(fid=fid[i], pos=pos[i], dims=dims)
endfor
end

The following steps outline the procedure for executing this example.

1. Save the routine to a file and place in the save_add directory.
2. Start ENVI.
3. To set the User Defined Move Routine to UD_MOVE_1, select File > Preferences.
4. Open a file and display a band.
5. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).
6. Move the zoom location, the move routine prints the data to the IDL log window.

**Example: Widget User Defined Move Routine**

This example creates a user defined move routine that outputs the current display number, zoom position and starting pixel values to a widget. The routine first checks for the text widget used to display the move routine data and if not found, creates one. The text string is then output to the text widget or display.

The sample code listing follows and can be found in the file irudmv2.pro in the lib directory of the installation.
ENVI Programmer's Guide

Chapter 6: Interactive User Routines

pro ud_move_2, dn, xloc, yloc, xstart=xstart, ystart=ystart
common ud_move_2_c, ud_wid, data

; Check for a valid widget id. If the widget ID is not valid then create the widget.
if (n_elements(ud_wid) eq 0) then ud_wid = -1L
if (widget_info(ud_wid, /valid) eq 0) then begin
    title = 'Custom Move Routine'
    envi_center, xoff, yoff
    base = widget_base(title=title, xoff=xoff, yoff=yoff, $ /row, group=envi_main_base())
    sb = widget_base(base, /column, /frame)
    sb1 = widget_base(sb, /col)
    lab = widget_label(sb1, value='Line Header Data')
    tw = widget_text(sb1, xs=40, ys=5)
    widget_control,base,/realize
    data = {tw:tw}
    ud_wid = base
endif

; Update the text widget with the current information.
; For now just display the dn, xloc, yloc, xstart, ystart.
msg = ['DN ' + string(dn), 'Loc (x,y): ' + $ string(xloc) + ', ' + string(yloc), 'Start (x,y): ' + $ string(xstart) + ', ' + string(ystart)]
widget_control, data.tw, set_value=msg, /no_copy
end

The following steps outline the procedure for executing this example.

1. Save the routine to a file and place in the save_add directory.
2. Start ENVI.
3. Set the User Defined Move Routine to UD_MOVE_2, select File > Preferences.
4. Open a file and display a band.
5. Move the zoom location, the move routine outputs the data to the displayed text widget.

Enhancements to this example would use a menu bar on the widget to provide additional controls. The controls could allow saving the displayed data to file, selecting a housekeeping filename or other similar operations.
Chapter 7:
Custom File Input

This chapter covers the following topics:

<table>
<thead>
<tr>
<th>Types of Image Ingest</th>
<th>154</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing Image File Headers</td>
<td>155</td>
</tr>
<tr>
<td>Custom Image Read Procedures</td>
<td>158</td>
</tr>
<tr>
<td>Spatial Read Procedure</td>
<td>160</td>
</tr>
<tr>
<td>Spectral Read Procedure</td>
<td>163</td>
</tr>
</tbody>
</table>
Types of Image Ingest

ENVI provides a very powerful interface for importing files not directly supported. In fact, many files can be opened in ENVI by interactively specifying the number of samples, lines, and bands, data type, header offset, and data storage interleave. The data in these files must be stored as BSQ, BIL or BIP. A more automated approach is to parse the header for the necessary parameters and use ENVI_SETUP_HEAD to open the file. The only custom development needed is to parse the header and open the file. Once the file is open, ENVI handles the remainder of the I/O.

When the files do not conform to any of the standard storage formats, users can still integrate the data without conversion. By creating a spatial and spectral read routine, the data quickly become integrated into ENVI. As the name implies, the spatial read routine handles all spatial data requests - from the whole image to a single pixel. The spectral read routine is responsible for all spectral requests. ENVI breaks down all input data requests into these two fundamental types, allowing easy integration of custom read routines.

Methods of importing files into ENVI are covered in more detail in the following sections.
To automatically import files currently not supported, users must develop a parsing routine to extract the basic file information. If the data are not stored as BSQ, BIL, or BIP, a custom read routine must also be developed (see “Custom Image Read Procedures” on page 158). At a minimum, the following information must be specified in order to open a file in ENVI.

- Number of samples, lines, and bands
- Data type
- Offset to image data
- File storage order, BSQ, BIL or BIP
- Byte storage order, Host (Intel), Network (IEEE)

Additional file information may also be parsed from the header, including any georeferencing information. The file is opened using ENVI_SETUP_HEAD (see “ENVI_SETUP_HEAD” on page 510) with the keywords set from the parsed header information. Make sure to set the OPEN keyword to open the file and optionally the WRITE keyword to write an ENVI header file.

The strategy for parsing the header depends on the header format. Some headers are keyword/value based and work well when read into a string variable. The STRPOS function can be used to locate the keyword, the following string value is converted to the appropriate type. Other headers use a fixed location and length for each header parameter. These headers are easily parsed using file positioning and single reads for each of the parameters.

The following examples illustrate parsing headers and opening files in ENVI.

**Example: Parsing a Keyword/Value Header**

This example illustrates how to parse a keyword/value header for the necessary file parameters. The keywords/values in this example are separated only by a space. Other header files may use a separator like an equal sign instead.

This example assumes a 512 byte header appended to the front of a BSQ byte image file with the following keywords.

```
SAMPLES value
LINES value
BANDS value
```
The sample code follows.

```plaintext
pro parse_header, fname
  buf = bytarr(512)
  openr, unit, fname, /get_lun
  readu, unit, buf
  free_lun, unit
  hdr = strupcase(string(buf))
  loc = strpos(hdr, 'SAMPLES') + 7
  ns = long(strmid(hdr, loc, strlen(hdr))
  loc = strpos(hdr, 'LINES') + 5
  nl = long(strmid(hdr, loc, strlen(hdr))
  loc = strpos(hdr, 'BANDS') + 5
  nb = long(strmid(hdr, loc, strlen(hdr)))
  envi_setup_head, fname=fname, ns=ns, nl=nl, nb=nb, $
  data_type=1, interleave=0, offset=512, /open
end
```

**Example: Parsing a Positional Header**

This example illustrates how to parse a header file with defined parameter positions. The file read position is set to the byte location of a particular value. The value is read according to the format of the header data. Typical formats include byte, integer, long, floating point, and double precision floating point numbers, as well as formatted ASCII data. This example reads binary header values for the samples, lines, and bands parameters.

This example assumes a 512 byte header appended to the front of a BSQ byte image file with the following values. All values are assumed to be in network (IEEE) storage order.

- bytes 20-23  - binary long number of samples
- bytes 30-33  - binary long number of lines
- bytes 40-43  - binary long number of bands

The sample code follows.

```plaintext
pro parse_header, fname
  openr, unit, fname, /get_lun
  ns = 0L
  nl = 0L
  nb = 0L
  point_lun, unit, 20L
  readu, unit, ns
  point_lun, unit, 30L
  readu, unit, nl
  point_lun, unit, 40L
  readu, unit, nb
  free_lun, unit
```
; Check to see if we need to swap
if (byte(256,0) eq 0) then begin
  byteorder, ns, /iswap
  byteorder, nl, /iswap
  byteorder, nb, /iswap
endif
envi_setup_head, fname=fname, ns=ns, nl=nl, nb=nb, $
  data_type=1, interleave=0, offset=512, /open
end
Custom Image Read Procedures

The ENVI read procedures provide a powerful mechanism for importing custom formats or files directly into ENVI, without the need for conversion. When files do not conform to any of the standard storage formats, it is necessary to create custom read procedures. By creating a spatial and spectral read routine, ENVI automatically integrates the data and makes them available to all ENVI functions. All spatial or spectral requests for data go through the specified read routines. In simple cases, read routines perform data format conversions; in more complex cases, they interface to an image database, where data requests are pulled directly from within the database.

A spatial read routine is responsible for all spatial data requests, ranging from a whole band to a single pixel. A spectral read routine handles all spectral data requests, ranging from a single spectrum to the spectra for an entire line. All input data requests are broken down into these two fundamental types. When opening files with custom read routines, use the READ_PROCEDURE keyword to ENVI_SETUP_HEAD (see “ENVI_SETUP_HEAD” on page 510) in order to define the spatial and spectral readers. Then, ENVI uses these procedures in place of its internal readers.

Files with custom readers must also have a defined file type in the filetype.txt file in the menu directory of the installation tree. Specifying a file type allows files to have an ENVI header, but to be opened by the custom open procedure which defines the READ_PROCEDURES. When the file has an ENVI header and is opened as an ENVI file, the header is read first. The header information is passed to the custom open routine using a PRE_FS keyword. The custom open routine parses the header in a normal manner and calls ENVI_SETUP_HEAD with the PRE_FS and the usual keywords set. A custom open procedure called OPEN_MYFILE is defined here:

```
PRO open_myfile, FNAME, CANCEL=CANCEL, R_FID=R_FID, PRE_FS=PRE_FS

where

FNAME - The filename including the path of the file to open.

CANCEL - Set this keyword to one when an error is encountered opening the file, otherwise set the keyword to zero.

R_FID - Set this keyword to the returned FID from ENVI_SETUP_HEAD.

PRE_FS - The value passed into the open procedure from parsing the ENVI header. This value must be passed directly to ENVI_SETUP_HEAD. If no ENVI header is present or the file is not opened as an ENVI file this value is undefined.
```
When opening custom files directly from the menu and not as ENVI files, the menu event handler first calls ENVI_PICKFILE (see “ENVI_PICKFILE” on page 479) to select the file. The filename is passed to the open routine as defined above. Supporting both methods of opening the files (directly or as ENVI files) allows files to remain in their native formats, with ENVI header parameter definitions, including map projections.

Custom read procedures often need special information not available through ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411). This is achieved by setting the INFO keyword to ENVI_SETUP_HEAD and extracting it’s handle value in the read routine using the H_INFO keyword to ENVI_FILE_QUERY. The associated INFO data are retrieved using the procedure HANDLE_VALUE with the H_INFO handle.

Spatial and spectral read routines are detailed in the following sections.
Spatial Read Procedure

Spatial read routines handle all spatial data requests ranging from an entire band to a single pixel. Spatial data are used extensively throughout ENVI, including display data, processing functions, and interactive routines. Read routines are not concerned with the originating source of the request; they must satisfy the data request.

Spatial requests specify X and Y starting and ending pixels and the desired band number. The input file unit number for the opened input file is also passed to the routine. Spatial read routines are defined as follows:

```pro
PRO myread_spatial, UNIT, R_DATA, FID, BAND, XS, YS, XE, YE,$
    _EXTRA=_EXTRA
```

where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIT</td>
<td>The file unit number already open for reading.</td>
</tr>
<tr>
<td>R_DATA</td>
<td>The parameter variable for the returned data. The read procedure must define this variable before exiting.</td>
</tr>
<tr>
<td>FID</td>
<td>The file id of the input image file.</td>
</tr>
<tr>
<td>BAND</td>
<td>The band position for the desired data. BAND is a long value ranging from zero to one minus the number of bands.</td>
</tr>
<tr>
<td>XS</td>
<td>The X starting pixel for the spatial request (in file coordinates).</td>
</tr>
<tr>
<td>YS</td>
<td>The Y starting pixel for the spatial request (in file coordinates).</td>
</tr>
<tr>
<td>XE</td>
<td>The X ending pixel for the spatial request (in file coordinates).</td>
</tr>
<tr>
<td>YE</td>
<td>The Y ending pixel for the spatial request (in file coordinates).</td>
</tr>
<tr>
<td>_EXTRA</td>
<td>This keyword must be specified to collect keywords not used by custom read routines.</td>
</tr>
</tbody>
</table>

*Table 7-1: Spatial Read Procedure Parameters*

The read routine reads the appropriate data from the file and stores the results in the R_DATA parameter. Opening and closing of input files is performed external; the corresponding file unit is passed as the UNIT parameter.
All read procedures should use the _EXTRA keyword to collect keywords not used by the custom read routines. The use of _EXTRA prevents errors when calling a routine with a keyword that is not listed in its definition.

The following example illustrates the use of a custom spatial read procedure.

**Example: Unsigned Integer Spatial Reader**

This example defines a spatial read routine used to simulate unsigned integer values.

- Data from 0 to 32767 are mapped from 0 to 32767
- Data from 32768 to 65535 are mapped from -32768 to -1

This routine takes the data and re-maps them into a continuous range from -38768 to 32767. Now the data are in the following ranges:

- Data from 0 to 32767 are mapped from -32768 to -1
- Data from 32768 to 65535 are mapped from 0 to 32767

Although the displayed data values are shifted, the images are displayed properly. The following band math expression would convert the images to long and allow display of the proper values:

```
long(b1) + 32768L
```

The spatial read routine uses ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411) to get necessary information about the image file. The output array is allocated as an integer array, since this read routine works only with integer data. Often, read routines need to support a variety of data types and must allocate arrays based on the file data type. Based on the file interleave, the unshifted data are placed into the output array. After completing the data ingest, the routine checks for byte swapping and performs the data shift. Using the following formula, the data are shifted to the simulated unsigned integer range.

```
(r_data + (r_data < 0) * 65536l) - 32768L
```

The sample code listing follows and can be found in the file fiunit.pro in the lib directory of the installation.

```pro
pro utest_spatial, unit, r_data, fid, band, xs, ys, xe, ye,$
   _extra=_extra
; Get the necessary file information
envi_file_query, fid, ns=ns, nl=nl, nb=nb, $
   interleave=interleave, offset=offset, $
   byte_swap=byte_swap
; Calculate the output size and allocate the array
o_ns = xe - xs + 1
o_nl = ye - ys + 1
```

r_data = intarr(o_ns, o_nl, /nozero)
; Read according to the file interleave
case interleave of
  0: begin
    a_offset = offset + 2 * (ns*nl*band + ys*ns) $
    a = assoc(unit, intarr(ns, /nozero), a_offset)
    for i=0L,o_nl-1 do r_data[0,i] = a[xs:xe, i]
  end
  1: begin
    aout = assoc(unit, intarr(ns, /nozero), offset)
    for i=ys,ye do r_data[0,i-ys] = $
      reform(aout[xs:xe, band+nb*i],/over)
  end
  2: begin
    aout = assoc(unit, intarr(nb, ns, /nozero), offset)
    for i=ys,ye do r_data[0,i-ys] = $
      reform(aout[band, xs:xe, i],/over)
  end
endcase
; check for byte swap
if (byte_swap) then byteorder, r_data
; Shift an unsigned data value to the top and bottom
r_data[0,0] = (r_data + (r_data lt 0) * 65536l) - 32768l
end

The spatial read routine composes half of a read procedure. The spectral read routine
is found in the “Example: Unsigned Integer Spectral Reader” on page 164. Both of
these routines can be found in the fiuint.pro in the ENVI lib directory. Save
this routine to a file and place in the save_add directory of the installation tree.
Instructions for completing the custom read routines are found following the section
on the Spectral Read procedure (see page 166). Adding the following line to the file’s
ENVI header defines the read procedures to use for accessing data.

    read procedures = {utest_spatial, utest_spectral}

Alternately, the READ_PROCEDURE keyword to ENVI_SETUP_HEAD could be
set as follows

    read_procedure = ['utest_spatial', 'utest_spectral']

Either technique sets the read procedure accordingly and simulates the unsigned
integer data. These read procedures only work with unsigned integer data and should
not be run on files with other data types.
Spectral Read Procedure

Spectral read routines handle all spectral data requests ranging from all spectra for an entire line to a single spectrum. Unlike spatial data requests, the maximum spectral requests are limited to a single line. Spectral data are used extensively throughout ENVI, for a number of processing functions and interactive routines.

Spectral requests specify the bands to read, starting and ending pixels, and the line. Spectral readers are also required to open and close the input files. Input filenames are returned from ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411) using the supplied FID. Spectral read routines are defined as follows.

```
PRO myread_spectral, FID, POS, XS=XS, XE=XE, Y=Y, 
SPECTRA=SPECTRA,$
_EXTRA=_EXTRA
```

where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>The file id of the input image file.</td>
</tr>
<tr>
<td>POS</td>
<td>The band positions for the desired spectra. POS is a long array with values ranging from zero to one minus the number of bands. The POS parameter is undefined when all bands are desired.</td>
</tr>
<tr>
<td>XS</td>
<td>The X starting pixel for the spectral request (in file coordinates).</td>
</tr>
<tr>
<td>XE</td>
<td>The X ending pixel for the spectral request (in file coordinates).</td>
</tr>
<tr>
<td>Y</td>
<td>The Y line number for the spectral request (in file coordinates).</td>
</tr>
<tr>
<td>SPECTRA</td>
<td>The name of the variable in which the requested spectra are to be stored.</td>
</tr>
<tr>
<td>_EXTRA</td>
<td>This keyword must be specified to collect keywords not used by custom read routines.</td>
</tr>
</tbody>
</table>

**Table 7-2: Spectral Read Procedure Parameters**

The read routine reads the appropriate spectra from the file and stores the results in the SPECTRA variable. Opening and closing of input files is performed within the
spectral read procedure. All read procedures should use the _EXTRA keyword to collect keywords not used by the custom read routines. The use of _EXTRA prevents errors when calling a routine with a keyword that is not listed in its definition. If POS is undefined, all the bands for the requested spectra are returned.

The following example further illustrates the use of a custom spectral read procedure.

**Example: Unsigned Integer Spectral Reader**

This example defines a spectral read routine used to simulate unsigned integer values.

- Data from 0 to 32767 are mapped from 0 to 32767
- Data from 32768 to 65535 are mapped from -32768 to -1

This routine takes the data an re-maps it into a continuous range from -38768 to 32767. Now the data are in the following ranges:

- Data from 0 to 32767 are mapped from -32768 to -1
- Data from 32768 to 64535 are mapped from 0 to 32767

Although the displayed data values are shifted the images display properly. The following band math expression would convert the images to long and allow display of the proper values.

```plaintext
long(b1) + 32768L
```

The spatial read routine uses ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411) to get necessary information about the image file and opens the input file. Based on the file interleave, the spectral data are read and saved in the output array SPECTRA. After completing the data ingest, the routine checks for byte swapping and performs the data shift. Using the following formula, the data are shifted to the simulated unsigned integer range.

```plaintext
(spectra + (spectra lt 0) * 65536L) - 32768L
```

The sample code listing follows and can be found in the file fiunit.pro in the lib directory of the installation.

```plaintext
pro utest_spectral, fid, pos, xs=xs, xe=xe, y=y, $spectra=spectra, _extra=extra
;
envi_file_query, fid, fname=fname, ns=ns, nl=nl, nb=nb, $offset=offset, interleave=interleave, $byte_swap=byte_swap
o_nb = n_elements(pos)
openr, unit, fname, /get_lun
case interleave of
0: begin
```
The spectral read routine composes half of a read procedure. The spatial read routine (described in “Example: Unsigned Integer Spatial Reader” on page 161) composes the other half. Save both of these routines to a file and place in the save_add directory of the installation tree. Adding the following line to the ENVI header defines the read procedures to use for accessing data.

```
read procedures = {'utest_spatial', 'utest_spectral'}
```

Alternately, when ENVI_SETUP_HEAD is called, the read procedures are defined by calling the READPROCEDURE keyword as follows:

```
read_procedure = ['utest_spatial', 'utest_spectral']
```

Either technique sets the read procedure accordingly and simulates the unsigned integer data.

**Note**

The example read procedures described here only work with unsigned integer data and should not be run on files with other data types.
The following steps outline the procedure for executing this example.

1. Save both read procedures to separate files and place the file in the save_add directory.

2. Edit the ENVI header of the integer data file using a text editor and add the following line:

   ```
   read procedures = {utest_spatial, utest_spectral}
   ```

3. Start ENVI.

4. Open the data file and display an image.

5. Use the Cursor Location/Value or interactive histogram to view the modified data range.
Chapter 8:
Additional Topics in ENVI Programming

This chapter covers the following topics:

- Coordinate Systems in ENVI .......... 168
- Regions of Interest ................... 171
- Using the Endmember Collection Widget . 182
- Programmatically Controlling ENVI Image Display Windows . ................. 185

- ENVI Installation Directory Components 187
- Basic ENVI Library Routines Reference . 190
- Quick Reference of ENVI Library Function Variable Codes . ................. 196
Coordinate Systems in ENVI

ENVI uses several different types of coordinate systems, some referring to the location of pixels within an image window, others to the location of image data within an array variable or file. In order to avoid confusion, both in ENVI programming and in common use of interactive ENVI, it is important to understand the differences among the various coordinate systems.

Image Coordinates (or Pixel Coordinates)

Image coordinates refer to the location of an image pixel in an ENVI Display window and are generic (sample, line) coordinates. Image coordinates are relatively simple in that they always increase (one unit for every pixel) with increasing sample and line number. For the samples coordinate, this will always be as you move to the right in an ENVI display window. The lines coordinate will increase from top to bottom.

The image coordinates for the first pixel in an image are defined by the XSTART and YSTART values in the image’s header file. For most images, ENVI sets the default XSTART and YSTART values to 1, defining the first pixel in an image with a coordinate of (1,1). Thus, if the image were an IDL 2D array variable, the data contained in subscript position [0,0] would correspond to image coordinate (1,1). If XSTART or YSTART are set to any other values (including negative numbers or zero) the image coordinates will begin incrementing from these values.

File Coordinates

File coordinates refer to the position of an image pixel within an IDL array and are equivalent to IDL array subscript positions. Unlike image coordinates, the file coordinates are always zero-based numbers, because IDL arrays are subscripted from zero to the number of elements minus one.

XSTART and YSTART

The use of the XSTART and YSTART values to define where the image coordinates begin, allow ENVI images to use a generic coordinate system that references an image other than itself. For example, if a small image is extracted as a spatial subset of a much larger image, the smaller, spatially subsetted image can retain its original image coordinates by setting its XSTART and YSTART to the first sample and line number of the subset. In some instances, ENVI processing routines automatically set the resulting image’s XSTART and YSTART values appropriately. For instance, when performing an Image-to-Image Registration, the registered image’s XSTART and YSTART values are set relative to the base image’s image coordinates. This
allows the two images to be directly compared using a common image coordinate system (as you can see when doing a dynamic overlay of the registered result and the base—the link offsets are computed directly from the XSTART and YSTART).

**Working with the XSTART and YSTART Programmatically**

If you were to use ENVI’s *File > Edit ENVI Header* to find an image’s X and YSTART values, the relationship between file coordinates and image coordinates is simply:

(sample) image coordinate = file coordinate + XSTART

(line) image coordinate = file coordinate + YSTART

However, it is important to recognize that the XSTART and YSTART values are reported differently within ENVI than they are in ENVI batch mode. In batch mode, the XSTART and YSTART values returned by ENVI_FILE_QUERY are reported as zero-based numbers; they are always one less than the values reported by the Header Info dialog in ENVI. For example, if a file has the standard XSTART and YSTART values of 1, then *File > Edit ENVI Header* reports the values as 1. Working with the same file in batch mode, ENVI_FILE_QUERY reports the XSTART and YSTART values as zero! Thus, in batch mode, the relationship between image coordinates and file coordinates becomes:

(sample) image coordinate = file coordinate + XSTART + 1

(line) image coordinate = file coordinate + YSTART + 1

The same relationship holds when defining a new file’s XSTART and YSTART values in batch mode using ENVI_SETUP_HEAD or ENVI_ENTER_DATA—the XSTART and YSTART values are defined as zero-based numbers. To make a file whose first pixel will have an image coordinate of (1,1) set the XSTART and YSTART to zero.

When coding a user function that processes and returns image data, if you allow the user to spatially subset the input image it will be important to keep track of the correct XSTART and YSTART values for the resulting output image (so that its image coordinates will correctly reference its original). The following code excerpt illustrates one method of updating the coordinates:

```plaintext
; select an input image and return the DIMS
; ENVI_SELECT, fid=fid, dims=dims...

; get the image's XSTART and YSTART values
; ENVI_FILE_QUERY, fid, xstart=xs, ystart=ys...
```
; if the starting sample and line are not zero
; then it was spatially subsetted so you'll
; need to update the XSTART and YSTART for the
; output image's header file
;
IF (dims[1] ne 0) THEN xs = xs + dims[1]
IF (dims[3] ne 0) THEN ys = ys + dims[3]

Because the batch mode XSTART and YSTART values are zero-based numbers, you
  can add them to the DIMS values which are in file coordinates. When the header for
  the processed file is written, the XSTART and YSTART values can be set using the
  XS and YS variables.

For additional examples of converting between image coordinates and file
  coordinates in an ENVI procedure, see “User Move Routines” on page 148.
Regions of Interest

Regions of Interest (ROIs), also referred to as Areas of Interest, are selected image subsets usually drawn by the user. These regions are often irregularly-shaped and typically used to extract statistics for classification, masking, and other operations. ENVI allows selection of any combination of polygons, vectors, or points as a ROI. Multiple ROIs can be defined for a single image and subsequently used with other images.

Processing with ROIs

Graphically, an ROI is a set of polygons, polylines, or points. However, from a processing standpoint, ROIs are addresses with associated data. Most ROI processing routines do not need the spatial correlation that they have graphically, but instead need the associated data. For example, the average spectrum from an ROI is calculated by summing the pixel values for each band and dividing by the total points in the ROI, regardless of where the points lie within the ROI. You do not need to know the addresses of each ROI point to calculate the mean.

ENVI provides a set of functions to work with ROIs. These functions allow for the opening of ROI files, the listing of all open ROIs, getting addresses, retrieving data, converting ids to DIMS pointers, and creating ROIs. When working with ROIs, there must be a tie between an ROI and a data file. This is performed by first selecting a file and then retrieving the list of ROIs that match the file’s spatial dimensions. The selected file does not have to be the one used to draw the ROI; it only has to have the same spatial dimensions. Next, the desired ROIs are selected from the returned list and used with the file information to get the associated data. The entire ROI data is returned in a single array (there is no tiling associated with ROIs).

Note

ROIs are related to a file by the spatial dimensions, number of samples, and number of lines. Use the reconcile ROI process to map an ROI to a file with different spatial dimensions.
Table 8-1 lists the available ENVI ROI routines.

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_CREATE_ROI</td>
<td>Create a new ROI</td>
<td>369</td>
</tr>
<tr>
<td>ENVI_DEFINE_ROI</td>
<td>Add objects to an ROI</td>
<td>376</td>
</tr>
<tr>
<td>ENVI_DELETE_ROIS</td>
<td>Delete ROIs from ENVI</td>
<td>379</td>
</tr>
<tr>
<td>ENVI_GET_ROI</td>
<td>Get the address of an ROI</td>
<td>435</td>
</tr>
<tr>
<td>ENVI_GET_ROI_DATA</td>
<td>Get the data associated with and ROI</td>
<td>436</td>
</tr>
<tr>
<td>ENVI_GET_ROI_DIMS_PTR</td>
<td>Convert the ROI id to a DIMS pointer value</td>
<td>438</td>
</tr>
<tr>
<td>ENVI_GET_ROI_IDS</td>
<td>Get a list of ROI ids</td>
<td>439</td>
</tr>
<tr>
<td>ENVI_SAVE_ROIS</td>
<td>Save ROIs in ENVI</td>
<td>496</td>
</tr>
<tr>
<td>ENVI_RESTORE_ROIS</td>
<td>Open and load a saved ROI file</td>
<td>493</td>
</tr>
</tbody>
</table>

Table 8-1: Region of Interest (ROI) routines available in ENVI.

Selecting ROIs

To understand the ROI selection, it is useful to remember that an ROI has an associated number of samples and lines. For a given spatial size, there may be any number of associated ROIs, including none. ENVI_GET_ROI_IDS returns all available ROIs associated with a given spatial dimension. There are three ways to specify the desired spatial dimension to ENVI_GET_ROI_IDS:

- Using the keywords NS and NL
- Using the keyword FID to match the spatial dimensions of the file specified by the FID
- Using the keyword DN to match the spatial dimension of the image in the display specified by DN

The compound widget WIDGET_MULTI (see “WIDGET_MULTI” on page 744) provides an excellent method for additional ROI selections. Once the final ROIs are
selected the user can use them to get associated ROI data, use ROIs as input into a processing function like statistics, or get the ROI addresses.

In addition to the ROI's drawn in the current ENVI session, any previously saved ROI files are loaded using ENVI_RESTORE_ROIS. All ROIs in the file are loaded. Although ENVI_RESTORE_ROIS does not return any ROI ids, the loaded ROIs are now available using ENVI_GET_ROI_IDS.

The following examples demonstrate ROI selection.

**Example: ROI Selection**

In this example, the basic ROI selection uses ENVI_GET_ROI_IDS (see “ENVI_GET_ROI_IDS” on page 439) with an associated spatial dimension. Three ROI are interactively drawn on an image. The image file is selected using ENVI_SELECT (see “ENVI_SELECT” on page 500) and the spatial dimensions are accessed using ENVI_FILE_QUERY (see “ENVI_FILE_QUERY” on page 411). The ROI ids are then retrieved by specifying the NS and NL for the file. For comparison the ROI ids are also retrieved by specifying the FID for the image file. The following steps outline this process:

1. Start ENVI, open a file, and display a grayscale image.
2. Interactively draw three separate polygon ROIs each with around 200 points.
3. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).
4. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to select the displayed file.
5. Select the displayed file (the grayscale).
   
   ```idl```
   `envi_select, title='Input Filename', fid=fid`
   ```idl```
6. Type the following on the ENVI command line to get the number of samples and lines.
   
   ```idl```
   `envi_file_query, fid, ns=ns, nl=nl`
   ```idl```
7. Type the following on the ENVI command line to get all the ROIs associated with a given number of samples and number of lines, print the result.
   
   ```idl```
   `roi_ids = envi_get_roi_ids(ns=ns, nl=nl)`
   `print, roi_ids`
8. Type the following on the ENVI command line to get all the ROIs with the same spatial dimensions as the file specified by FID, then print the results.

```python
roi_ids = envi_get_roi_ids(fid=fid)
print, roi_ids
```

The two printed ROI_IDS should be the same since they reference the same spatial dimensions.

**Example: ROI Selection and WIDGET_MULTI**

This example expands on the example “Example: ROI Selection” on page 173 by using WIDGET_MULTI (see “WIDGET_MULTI” on page 744) for additional ROI selection. The interactive steps have been added into a procedure to select a group of ROIs.

This example first selects a file to use as the spatial dimension reference for the ROI selection. ENVI_GET_ROI_IDS (see “ENVI_GET_ROI_IDS” on page 439) returns the list of ROI ids and the optional keyword ROI_NAMES returns the associated ROI names. If no ROIs are found then a single element array with the value -1 is returned and the routine prints an error message and exits. An auto-managed widget is created for final ROI selection. The selected items from the list are indicated by the variable PTR and the ROI names and ids are printed.

The sample code listing follows and can be found in the file ufoi1.pro in the lib directory of the installation.

```plaintext
pro roi_multi_sel
  envi_select, title='Input Filename', fid=fid
  if (fid eq -1) then return
  roi_ids = envi_get_roi_ids(fid=fid, $roi_names=roi_names)
  if (roi_ids[0] eq -1) then begin
    print, 'No regions associated with the selected file'
    return
  endif
  ; Compound widget for ROI selection
  base = widget_auto_base(title='ROI Selection')
  wm = widget_multi(base, list=roi_names, uvalue='list', /auto)
  result = auto_wid_mng(base)
  if (result.accept eq 0) then return
  ptr = where(result.list eq 1)
  print, roi_names[ptr]
  print, roi_ids[ptr]
end
```

1. Save the procedure to a file and place the save_add directory.
2. Start or restart ENVI.
3. Open a file and display a grayscale image.
4. Interactively draw three separate polygon ROIs each with around 200 points.
5. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).
6. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to run the routine.
   
   `roi_multi_sel`

   The sample code in this example can be used as a model for users wishing to allow ROI selection in their user functions.

**Example: Restore Saved ROIs**

This example interactively restores a saved ROI file. The compound widget ENVI_PICKFILE (see “ENVI_PICKFILE” on page 479) is used to select the file and the returned filename is then passed to ENVI_RESTORE_ROIS (see “ENVI_RESTORE_ROIS” on page 493). An informational message is displayed showing the restored ROIs.

1. Start ENVI.
2. Open a file and display a grayscale image.
3. Interactively draw three separate polygon ROIs each with around 200 points.
4. Save the ROIs to a file.
5. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).
6. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to select the ROI file you saved above.

   ```python
   name = envi_pickfile(title='ROI File', filter='*.roi')
   ```

7. Type the following on the ENVI command line to restore the saved ROIs.

   ```plaintext
   envi_restore_rois, name
   ```

   Now two sets of the same ROIs are loaded into ENVI - the ones created interactively and the ones loaded from the file. Although these two sets originated from the same source, they are now considered independent.
Using ROI Data

Once an ROI is selected, acquiring and processing the data is quite simple. The ROI data is returned in an array using `ENVI_GET_ROI_DATA`. The file from which to extract the data is defined by the FID variable passed into `ENVI_GET_ROI_DATA`. The following examples build on the previous ROI selection examples by adding the data request.

Example: Using ROI Data

The basic ROI selection uses `ENVI_GET_ROI_IDS` (see “ENVI_GET_ROI_IDS” on page 439) with an associated spatial dimension. This example interactively selects all ROIs associated with the spatial dimensions specified by FID. The ROI data from the first band is accessed using `ENVI_GET_ROI_DATA` (see “ENVI_GET_ROI_DATA” on page 436). The ROI mean for this band is calculated and printed.

1. Start ENVI, open a file, and display a grayscale image.
2. Interactively draw one polygon ROI with around 200 points.
3. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).
4. Type the following on the ENVI command line (the line that contains the prompt “ENVI>”) to select the displayed file. Select the displayed file (the grayscale).
   ```
   envi_select, title='Input Filename', fid=fid, pos=pos
   ```
5. Type the following on the ENVI command line to get all the ROIs with the same spatial dimensions as the file specified by FID.
   ```
   roi_ids = envi_get_roi_ids(fid=fid)
   ```
6. Type the following on the ENVI command line to get the ROI data for the first band and then calculate the ROI mean value.
   ```
   data = envi_get_roi_data(roi_ids[0], fid=fid, pos=pos[0])
   print, 'ROI mean = ', total(data) / n_elements(data)
   ```

To access any remaining bands in the POS array `ENVI_GET_ROI_DATA` is called again with the POS keyword set to the next element. This process continues in a loop until all bands have been accessed. Or, the ROI data from all of the bands in the file can be extracted in a single call to `ENVI_GET_ROI_DATA`. This interactive example is for demonstration purposes only. In practice these steps are part of a user function.
Example: Calculating ROI Means

This example extends the “Example: ROI Selection and WIDGET_MULTI” on page 174 to request ROI data and calculate the ROI mean.

This example uses ENVI_SELECT to selects the file to use as the spatial dimension reference. Then ENVI_GET_ROI_IDS is used to return the list of ROI ids and the keyword ROI_NAMES returns the associated ROI name. If no ROIs are found then a single element array with the value -1 is returned and the routine prints an error message and exits. Next, an auto-managed widget is created for the final ROI selection. The selected items from the list are indicated by the variable PTR. The data for each band of a selected ROI is read in and the mean value is computed. The resulting mean values are printed.

The sample code listing follows and can be found in the file ufroi2.pro in the lib directory of the installation.

```pro
roi_mean
  envi_select, title='Input Filename', fid=fid, pos=pos
  if (fid eq -1) then return
  roi_ids = envi_get_roi_ids(fid=fid, roi_names=roi_names)
  if (roi_ids[0] eq -1) then begin
    print, 'No regions associated with the selected file'
    return
  endif
  ; Compound widget for ROI selection
  base = widget_auto_base(title='ROI Selection')
  wm = widget_multi(base, list=roi_names, uvalue='list', /auto)
  result = auto_wid_mng(base)
  if (result.accept eq 0) then return
  ptr = where(result.list eq 1, count)
  result = dblarr(n_elements(pos))
  ; ROI Mean calculation
  for i=0L count-1 do begin
    for j=0L, n_elements(pos)-1 do begin
      data = envi_get_roi_data(roi_ids[ptr[i]], fid=fid, $
        pos=pos[j])
      result[j] = total(data, /double) / n_elements(data)
    endfor
    print, roi_names[ptr[i]]
    print, result
  endfor
end
```

1. Save the procedure to a file and place in the save_add directory.
2. Start or restart ENVI.
3. Open a file and display a grayscale image.
4. Interactively draw three separate polygon ROIs each with around 200 points.

5. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).

6. Type the following on the ENVI command line to run the routine.

   `roi_mean`

The sample code in this example is a useful model for adding ROI selection to user functions. To calculate ROI statistics users should use ENVI_STATS_DOIT (see “ENVI_STATS_DOIT” on page 519) and set the ROI DIMS pointer, see “Using ROI DIMS Pointers” in the following section.

### Using ROI DIMS Pointers

A few routines in ENVI allow spatial subsets using an ROI. In these cases the first element of the DIMS array is a pointer to the desired ROI. The ENVI routine ENVI_GET_ROI_DIMS_PTR converts an ROI id into a DIMS pointer.

**Note**

An ROI DIMS pointer value can change when ROIs are added or deleted. It is best to convert the ROI ids when defining the DIMS array.

### Example: ROI DIMS Pointer

This example converts the “Example: Calculating ROI Means” on page 177 to use ENVI_STATS_DOIT (see “ENVI_STATS_DOIT” on page 519) to calculate the ROI statistics. ENVI_STATS_DOIT is one of many batch processing routines available within ENVI, see Chapter 9, “ENVI Routines” for details.

This example first selects the image file used as the spatial reference for the ROIs. Next, ENVI_GET_ROI_IDS (see “ENVI_GET_ROI_IDS” on page 439) is used to return the list of ROI ids and the keyword ROI_NAMES returns the associated ROI name. If no ROIs are found then a single element array with the value -1 is returned and the routine prints an error message and exits. Now an auto-managed widget is created for the final ROI selection. The selected items from the list are indicated by the variable PTR. The basic statistics for each selected ROI are calculated using ENVI_STATS_DOIT. The resulting min, max, mean, and standard deviation vectors are printed.
The sample code listing follows and can be found in the file ufroi3.pro in the lib directory of the installation.

```idl
pro roi_stat
  envi_select, title='Input Filename', fid=fid, pos=pos
  if (fid eq -1) then return
  roi_ids = envi_get_roi_ids(fid=fid, roi_names=roi_names)
  if (roi_ids[0] eq -1) then begin
    print, 'No regions associated with the selected file'
    return
  endif
  ; Compound widget for ROI selection
  base = widget_auto_base(title='ROI Selection')
  wm = widget_multi(base, list=roi_names, uvalue='list', /auto)
  result = auto_wid_mng(base)
  if (result.accept eq 0) then return
  ptr = where(result.list eq 1, count)
  result = dblarr(n_elements(pos))
  ; ROI Stats calculation
  for i=0L, count-1 do begin
    dims = [envi_get_roi_dims_ptr(roi_ids[ptr[i]]), 0,0,0,0]
    envi_stats_doit, fid=fid, dims=dims, pos=pos, comp_flag=0, $
    report_flag=0, mean=mean, stdv=stdv, dmin=dmin, dmax=dmax
    print, roi_names[ptr[i]]
    print, dmin, dmax, mean, stdv
  endfor
end
```

1. Save the procedure to a file and place in the save_add directory.
2. Start or restart ENVI.
3. Open a file and display a grayscale image.
4. Interactively draw three separate polygon ROIs each with around 200 points.
5. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).
6. Type the following on the ENVI command line to run the routine.

   ```idl
   roi_stat
   ```

The sample code in this example is a useful model for calculating statistics for ROIs.
Using ROI Addresses

ENVI provides access to the spatial location of any ROI by using it’s addresses. ROI addresses are single element addresses referenced from the first pixel in the image and increasing in the sample direction. Table 8-2 shows some sample ROI addresses and their corresponding pixel for an image with ten samples and eight lines.

<table>
<thead>
<tr>
<th>ROI Address</th>
<th>Corresponding pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>First pixel in the image</td>
</tr>
<tr>
<td>12</td>
<td>The third pixel on the second line</td>
</tr>
<tr>
<td>30</td>
<td>The first pixel on the third line</td>
</tr>
<tr>
<td>79</td>
<td>The last pixel in the image</td>
</tr>
</tbody>
</table>

*Table 8-2: Sample ROI address for a 10x8 image.*

The routine ENVI_GET_ROI returns the address associated with the specified ROI. If necessary the address can be converted to spatial X and Y values, see the following example. ROI addresses are zero based, the first pixel in the image has the address of “0.”

**Example: ROI Addresses**

This example computes the X and Y values for a ROI address. The ROI selection uses ENVI_GET_ROI_IDS (see “ENVI_GET_ROI_IDS” on page 439) with an associated spatial dimension using the keyword FID. The FID is returned using the compound widget ENVI_SELECT (see “ENVI_SELECT” on page 500). Next, the ROI address returned from ENVI_GET_ROI (see “ENVI_GET_ROI” on page 435) are converted to (X,Y) locations and printed.

1. Start ENVI.
2. Open a file and display a grayscale image.
3. Interactively draw one polygon ROI with around 200 points.
4. Open the IDL development environment (PC, Macintosh) or the shell window ENVI was started in (UNIX).
5. Type the following on the ENVI command line to select the displayed file.

   ```idl```
   envi_select, title='Input Filename', fid=fid
   ```idl```
6. Type the following on the ENVI command line to get all the ROIs with the same spatial dimensions as the file specified by FID.
   
   ```python
   roi_ids = envi_get_roi_ids(fid=fid)
   ```

7. Type the following on the ENVI command line to get the ROI address.
   
   ```python
   addr = envi_get_roi(roi_ids[0])
   ```

8. Get the number of samples and lines in the file specified by FID.
   
   ```python
   envi_file_query, fid, ns=ns, nl=nl
   ```

9. Calculate and print the (X,Y) location for each point in the ROI
   
   ```python
   y = addr / ns
   x = addr - y * ns
   print, x
   print, y
   ```

This interactive example is for demonstration purposes only. In practice these steps are part of a user function.
Using the Endmember Collection Widget

The Endmember Collection dialog is one of ENVI’s most sophisticated compound widgets. It is used to collect training sets and endmembers for classification and mapping routines and can be incorporated into user functions with the routine ENVI_COLLECT_SPECTRA. However, this widget functions differently than all of the other ENVI widgets and requires some special instruction. (For users that are familiar with IDL widget programming, using the Endmember Collection widget is essentially no different than writing a customized event handler for the widget’s “Apply” button).

Figure 8-1: Endmember Collection widget

When a widget interface for a user function is auto-managed by ENVI, the GUI is always modal. That is, the widget interface will block the rest of ENVI from responding until the user selects the “OK” or “Cancel” button, either of which destroys the GUI (one destroys and proceeds, the other destroys and returns). In this mode, ENVI handles all of the widget events and conveniently returns the user input to the programmer in a structure variable. Using ENVI to auto-manage the GUI in this fashion allows user function procedures to take a very simple form, where they can execute linearly from the beginning of the file to the end.

Most built-in ENVI processing routines that have a user interface work similarly—they are modal and get destroyed after the user selects either “OK” or “Cancel.”
Unlike these GUIs, the Endmember Collection Dialog is not modal. After this dialog widow is displayed, all of ENVI’s menus remain active. The primary reason that this widget is not modal is that it serves as a repository of endmember spectra that can be used for a wide variety of routines. Thus, it makes sense to allow the user continued access to it after the original processing is finished. However, while the non-modality of this widget makes it more functional, it also makes it more complicated to use.

When writing a user function that requires endmember spectra to be selected for use in the processing routine, ENVI_COLLECT_SPECTRA can be used to display the Endmember Collection dialog. However, when the user clicks the “Apply” button, instead of returning the data collected by the widget to the programmer in a structure variable, ENVI automatically calls a second, completely separate, procedure into which the endmember data is passed. The name of the procedure that is called is defined by the PROCEDURE keyword to ENVI_COLLECT_SPECTRA. Thus, a user function that uses this widget is broken into two parts. The first procedure is the one defined in the menu file and is executed when the user selects the user function’s button from the ENVI menu. The first procedure ends after it calls ENVI_COLLECT_SPECTRA. The second procedure is executed when the ENVI user clicks on the “Apply” button and uses the information collected by the widget to carry out the user function’s processing. Because the Endmember Collection dialog will remain open, the ENVI user can modify the contents of the dialog window and run the second part of the user function multiple times by clicking on the “Apply” button.

In IDL widget programming, this second procedure is referred to as an event handler because it handles the widget event that occurs when the “Apply” button is pressed. This event-handler procedure can have any name, but must have the following keywords defined in its procedure definition statement:

```
pro MY_EVENT_HANDLER, fid=fid, pos=pos, dims=dims, "$ spec=spec, snames=snames, scolors=scolors, _extra=extra
```

The FID, POS and DIMS variables are passed into ENVI_COLLECT_SPECTRA when it is called; SPEC, S NAMES, and SCOLORS are collected by the Endmember Collection widget. The _EXTRA keyword is allowed as a means to pass additional information from the first procedure (the one that called ENVI_COLLECT_SPECTRA) to the event-handler procedure.

The two procedures that make up the single-user function are frequently included in the same file. When this organization is used, the first procedure (i.e., the one defined in the menu file) must be the last procedure in the file, otherwise IDL will not auto-compile both procedures when it starts ENVI. The file that contains the user function will have the following structure:
pro MY_EVENT_HANDLER, fid=fid, pos=pos, dims=dims, spec=spec, snames=snames, scolors=scolors, _extra=extra
   do the user function processing...
end

pro MY_USER_FUNCTION, event
   ENVI_SELECT, fid=fid, pos=pos, dims=dims
   info = {structure variable of useful information}
   do any pre-processing if necessary
   ENVI_COLLECT_SPECTRA, dims=dims, fid=fid, pos=pos,$
      title=title, procedure='my_event_handler',$
      h_info=info
end
Programmatically Controlling ENVI Image Display Windows

ENVI users who write user functions may wish to programmatically retrieve information about an ENVI display. This section discusses topics related to retrieving ENVI display information.

Getting the Window ID for Scroll, Image and Zoom Windows

The $W1$ keyword to the ENVI\_DISP\_QUERY routine can be used to find the window IDs for the Image, Zoom and Scroll windows of an ENVI display. This window ID is not the draw widget ID for the display windows. It is the value of !d.window, which can be used with IDL’s WSET or WSHOW routines.

Getting the Zoom factor of the Zoom Window and the Resize Factor of the Scroll Window

Both the zoom and resize factors can be obtained using keywords to the ENVI\_DISP\_QUERY routine. Use the ZFACT keyword to get the zoom factor and the REBIN keyword to get the resize factor.

Getting the Position of the Current Pixel

The ENVI routine, DISP\_GET\_LOCATION, can be used to return the x and y locations of the current pixel for a given ENVI display. For more information, see “DISP\_GET\_LOCATION” on page 312.

Moving the Zoom Window to a New Pixel Location

The ENVI routine, DISP\_GOTO, can be used to move the current pixel of a given display. The Zoom window will also be moved so that it is centered around the new current pixel location. If the new current pixel location is not within the current Image window, the Image and Scroll windows will also be moved so that they include the new current pixel. For more information, see “DISP\_GOTO” on page 314.

Closing an ENVI Image Display Group

If you need to close all three windows of an ENVI Image Display Group you actually have to fool ENVI into thinking that the user closed the window using the mouse. To do this you must define a widget event structure with the correct instructions, and
send the event to the event handler. The following code example illustrates how this
works. All you will have to do is specify the display number for the image display
you want to close, using the DN variable:

```python
id = WIDGET_BASE(uvalue = 'close')
top = WIDGET_BASE(uvalue = DN)
disp_event,{id:id, top:top}
```
ENVI Installation Directory Components

After installing ENVI, you will find an ENVIxx directory containing several subdirectories. Some of these subdirectories contain files that are particularly important for ENVI programming. The various files found in the ENVI installation directory are defined in the following tables (based on ENVI version 3.5).

**ENVI Subdirectories**

<table>
<thead>
<tr>
<th>Subdirectory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td><code>envi.run</code> (a special ASCII file used by IDL to auto-start ENVI)</td>
</tr>
<tr>
<td>data</td>
<td>ENVI’s default data directory (contains example data distributed with ENVI)</td>
</tr>
<tr>
<td>docs</td>
<td>PDF versions of the ENVI documentation set</td>
</tr>
<tr>
<td>filt_func</td>
<td>Filter function spectral library files (for resampling to known sensors)</td>
</tr>
<tr>
<td>help</td>
<td>ENVI Online Help and HTML versions of the Tutorials</td>
</tr>
<tr>
<td>lib</td>
<td>Example ENVI code (sample IDL procedures using some of the ENVI routines)</td>
</tr>
<tr>
<td>map_proj</td>
<td>Files that store the data related to ENVI map projection</td>
</tr>
<tr>
<td>menu</td>
<td>A variety of setup and configuration files (including the menu definition files)</td>
</tr>
<tr>
<td>save</td>
<td>The ENVI binary files (IDL <code>.sav</code> files)</td>
</tr>
<tr>
<td>save_add</td>
<td>Recommended storage location for user-written code (i.e., User Functions)</td>
</tr>
<tr>
<td>spec_lib</td>
<td>ENVI spectral library files</td>
</tr>
</tbody>
</table>

*Table 8-3: ENVI Subdirectories.*
The Menu Directory

All files in the menu directory are editable ASCII files related to the ENVI configuration and working environment. A list of these files is provided in Table 8-4.

<table>
<thead>
<tr>
<th>File</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>colors.txt</td>
<td>Files which define the ENVI graphics colors</td>
</tr>
<tr>
<td>colors25.txt</td>
<td></td>
</tr>
<tr>
<td>colors50.txt</td>
<td></td>
</tr>
<tr>
<td>display.men</td>
<td>Files which define ENVI menu buttons—display.men defines the Main Image Display menu and envi.men defines the main ENVI menu</td>
</tr>
<tr>
<td>envi.men</td>
<td></td>
</tr>
<tr>
<td>endmember_mapping_wizard.txt</td>
<td>File which contains the text for the Spectral Mapping Wizard</td>
</tr>
<tr>
<td>envi.cfg</td>
<td>The ENVI configuration file (updated when setting ENVI preferences)</td>
</tr>
<tr>
<td>e_locate.pro</td>
<td>A dummy .pro file used to ensure that the IDL path includes the menu directory</td>
</tr>
<tr>
<td>filetype.txt</td>
<td>File which defines the ENVI file types and their corresponding integer codes</td>
</tr>
<tr>
<td>sensor.txt</td>
<td>File which defines the ENVI sensor types and their corresponding integer codes</td>
</tr>
<tr>
<td>special_menu_buttons.txt</td>
<td>File for foreign language translation purposes</td>
</tr>
<tr>
<td>splash.tif</td>
<td>The TIFF image used as the ENVI splash screen (displayed on startup)</td>
</tr>
<tr>
<td>useradd.txt</td>
<td>File which contains the definitions for user added items (e.g., units, plot functions)</td>
</tr>
<tr>
<td>usersym.txt</td>
<td>File which contains user defined vector symbols</td>
</tr>
</tbody>
</table>

Table 8-4: Files found in the ENVI Menu directory.
The Map_Proj Directory

All files in the ENVI map_proj directory are editable ASCII files related to map projections.

<table>
<thead>
<tr>
<th>File</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>convert.txt</td>
<td>Contains example IDL procedures for converting between ASCII data and EVFs</td>
</tr>
<tr>
<td>datum.txt</td>
<td>File that contains all of ENVI's data related to supported map datums</td>
</tr>
<tr>
<td>ellipse.txt</td>
<td>File that contains all of ENVI's data related to supported map ellipsoids</td>
</tr>
<tr>
<td>map_proj.txt</td>
<td>File that contains all of ENVI's defined projection data</td>
</tr>
<tr>
<td>sp_nad27.txt</td>
<td>Files which contain all of the data associated with NAD27 and NAD38 State Plane projections (i.e., zone names and numbers)</td>
</tr>
<tr>
<td>sp_nad83.txt</td>
<td></td>
</tr>
</tbody>
</table>

Table 8-5: Files found in the map_proj directory.
Basic ENVI Library Routines Reference

Chapter 9, “ENVI Routines” provides a complete reference page for each library routine in alphabetical order. However, it’s often more useful to see related routines grouped together by functional area. Table 8-6 includes the basic ENVI library routines.

<table>
<thead>
<tr>
<th>Functional Area</th>
<th>Routine Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Mode</td>
<td>ENVI</td>
<td>Restores core ENVI save files when used with the RESTORE_BASE_SAVE_FILES keyword</td>
</tr>
<tr>
<td></td>
<td>ENVI_BATCH_INIT</td>
<td>Initiates a batch mode session</td>
</tr>
<tr>
<td></td>
<td>ENVI_BATCH_EXIT</td>
<td>Exits a batch mode session</td>
</tr>
<tr>
<td></td>
<td>ENVI_BATCH_STATUS_WINDOW</td>
<td>Enable and disable the batch status window</td>
</tr>
<tr>
<td>File Input &amp; Management</td>
<td>ENVI_PICKFILE</td>
<td>Widget for selecting a filename from disk</td>
</tr>
<tr>
<td>File Info</td>
<td>ENVI_SELECT</td>
<td>Widget for choosing an opened file</td>
</tr>
<tr>
<td></td>
<td>ENVI_OPEN_FILE</td>
<td>Opens a file from disk</td>
</tr>
<tr>
<td></td>
<td>ENVI_FILE_MNG</td>
<td>Closes and/or deletes files</td>
</tr>
<tr>
<td></td>
<td>ENVI_GET_FILE_IDS</td>
<td>Returns all opened file IDs</td>
</tr>
<tr>
<td></td>
<td>ENVI_OPEN_DATA_FILE</td>
<td>Opens external (non-ENVI) image files</td>
</tr>
<tr>
<td></td>
<td>ENVI_FILE_QUERY</td>
<td>Returns info from an ENVI header file</td>
</tr>
<tr>
<td></td>
<td>ENVI_FILE_TYPE</td>
<td>Translates between file type codes and descriptions</td>
</tr>
<tr>
<td></td>
<td>ENVI_SENSOR_TYPE</td>
<td>Translates between sensor type codes and descriptions</td>
</tr>
<tr>
<td></td>
<td>ENVI_SET_INHERITANCE</td>
<td>Return the ENVI inheritance structure</td>
</tr>
</tbody>
</table>

Table 8-6: Basic ENVI Routines Grouped by Functional Area
### Functional Area

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_ENTER_DATA</td>
<td>Enters an image into memory as an ENVI file</td>
</tr>
<tr>
<td>ENVI_SETUP_HEAD</td>
<td>Writes an ENVI header file for an image file</td>
</tr>
<tr>
<td>CF_DOIT</td>
<td>Creates a new ENVI image from existing FIDs</td>
</tr>
<tr>
<td>SLICE_DOIT</td>
<td>Outputs a vertical or horizontal data slice to file</td>
</tr>
<tr>
<td>ENVI_OUTPUT_TO_EXTERNAL_FORMAT</td>
<td>Outputs a file to a non-ENVI (external) format</td>
</tr>
</tbody>
</table>

### Reading Image Data (non-tiled)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_GET_DATA</td>
<td>Returns spatial image data from file</td>
</tr>
<tr>
<td>ENVI_GET_SLICE</td>
<td>Returns spectral image data from file</td>
</tr>
<tr>
<td>ENVI_GET_IMAGE</td>
<td>Returns stretched data from a display window</td>
</tr>
</tbody>
</table>

### ROI Processing

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_RESTORE_ROIS</td>
<td>Restores ROIs from file</td>
</tr>
<tr>
<td>ENVI_GET_ROI_ID</td>
<td>Returns available ROI Ids</td>
</tr>
<tr>
<td>ENVI_GET_ROI</td>
<td>Returns ROI pixel addresses as 1-D subscripts</td>
</tr>
<tr>
<td>ENVI_GET_ROI_DATA</td>
<td>Returns image data associated with an ROI</td>
</tr>
<tr>
<td>ENVI_GET_ROI_DIMS_PTR</td>
<td>Returns ROI pointer for use with DIMS variable</td>
</tr>
<tr>
<td>ENVI_CREATE_ROI</td>
<td>Returns a new ROI ID</td>
</tr>
<tr>
<td>ENVI_DEFINE_ROI</td>
<td>Adds pixels to an ROI</td>
</tr>
<tr>
<td>ENVI_SAVE_ROIS</td>
<td>Saves ROIs to disk</td>
</tr>
<tr>
<td>ENVI_DELETE_ROIS</td>
<td>Deletes ROI from the ENVI session</td>
</tr>
</tbody>
</table>

*Table 8-6: Basic ENVI Routines Grouped by Functional Area (Continued)*
<table>
<thead>
<tr>
<th>Functional Area</th>
<th>Routine Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tiling</strong></td>
<td>ENVI_INIT_TILE</td>
<td>Initiates tiled processing and returns the tile ID</td>
</tr>
<tr>
<td></td>
<td>ENVI_GET_TILE</td>
<td>Returns one spatial or spectral tile of image data</td>
</tr>
<tr>
<td></td>
<td>ENVI_TILE_DONE</td>
<td>Ends tiled processing</td>
</tr>
<tr>
<td><strong>Reporting</strong></td>
<td>ENVI_REPORT_INIT</td>
<td>Both initializes and ends status reporting</td>
</tr>
<tr>
<td></td>
<td>ENVI_REPORT_INC</td>
<td>Computes the status reporting increment</td>
</tr>
<tr>
<td></td>
<td>ENVI_REPORT_STAT</td>
<td>Updates the status reporting widget dialog</td>
</tr>
</tbody>
</table>

*Table 8-6: Basic ENVI Routines Grouped by Functional Area (Continued)*
## Table 8-6: Basic ENVI Routines Grouped by Functional Area (Continued)

<table>
<thead>
<tr>
<th>Functional Area</th>
<th>Routine Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Information</td>
<td>ENVI_GET_PROJECTION</td>
<td>Returns an ENVI_PROJ_STRUCT variable</td>
</tr>
<tr>
<td></td>
<td>ENVI_PROJ_CREATE</td>
<td>Builds a projection’s ENVI_PROJ_STRUCT</td>
</tr>
<tr>
<td></td>
<td>ENVI_MAP_INFO_CREATE</td>
<td>Creates a new ENVI_MAP_STRUCT variable</td>
</tr>
<tr>
<td></td>
<td>ENVI_ADD_PROJECTION</td>
<td>Adds a new projection to the ENVI session</td>
</tr>
<tr>
<td></td>
<td>ENVI_TRANSLATE_PROJECTION_UNITS</td>
<td>Translates between projection unit codes &amp; names</td>
</tr>
<tr>
<td></td>
<td>ENVI_TRANSLATE_PROJECTION_NAME</td>
<td>Translates between projection type codes &amp; names</td>
</tr>
<tr>
<td></td>
<td>ENVI_PROJ_READ_PROJ</td>
<td>Returns a variable containing all projection data</td>
</tr>
<tr>
<td></td>
<td>ENVI_PROJ_READ_ELLIPSE</td>
<td>Returns a variable containing all ellipse data</td>
</tr>
<tr>
<td></td>
<td>ENVI_PROJ_READ_DATUM</td>
<td>Returns a variable containing all datum data</td>
</tr>
<tr>
<td></td>
<td>ENVI_CONVERT_FILE_COORDINATES</td>
<td>Converts between file and map coordinates</td>
</tr>
<tr>
<td></td>
<td>ENVI_CONVERT_PROJECTION_COORDINATES</td>
<td>Converts coordinates between map projections</td>
</tr>
<tr>
<td></td>
<td>ENVI_CONVERT_FILE_MAP_PROJECTION</td>
<td>Converts a georeferenced file into a new projection</td>
</tr>
<tr>
<td>Functional Area</td>
<td>Routine Name</td>
<td>Usage</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>ENVI Widgets</td>
<td>AUTO_WID_MNG</td>
<td>Returns user-inputted info from an ENVI GUI</td>
</tr>
<tr>
<td></td>
<td>ENVI_PICKFILE</td>
<td>ENVI’s standard pickfile widget</td>
</tr>
<tr>
<td></td>
<td>ENVI_SELECT</td>
<td>ENVI’s standard file selection dialog</td>
</tr>
<tr>
<td></td>
<td>WIDGET_AUTO_BASE</td>
<td>Creates a widget base for auto-managed events</td>
</tr>
<tr>
<td></td>
<td>ENVI_CENTER</td>
<td>Returns centering offsets for widget positioning</td>
</tr>
<tr>
<td></td>
<td>WIDGET_SUBSET</td>
<td>ENVI’s standard Spatial Subset widgets</td>
</tr>
<tr>
<td></td>
<td>WIDGET_PROJ</td>
<td>ENVI’s standard projection selection widget</td>
</tr>
<tr>
<td></td>
<td>WIDGET_GEO</td>
<td>Widget for entering lat/lon data</td>
</tr>
<tr>
<td></td>
<td>WIDGET_MAP</td>
<td>Widget for editing map information</td>
</tr>
<tr>
<td></td>
<td>WIDGET_MENU</td>
<td>Widget for making radio and checkmark buttons</td>
</tr>
<tr>
<td></td>
<td>WIDGET_PMENU</td>
<td>Widget for making a pull-down menu</td>
</tr>
<tr>
<td></td>
<td>WIDGET_MULTI</td>
<td>Widget for selecting multiple items from a list</td>
</tr>
<tr>
<td></td>
<td>WIDGET_PARAM</td>
<td>Widget for entering a numeric parameter</td>
</tr>
<tr>
<td></td>
<td>WIDGET_TOGGLE</td>
<td>Creates an ENVI toggle button</td>
</tr>
<tr>
<td></td>
<td>WIDGET_SLIDER</td>
<td>Widget for defining a value with a slider</td>
</tr>
<tr>
<td></td>
<td>WIDGET_RGB</td>
<td>Widget for editing color values</td>
</tr>
<tr>
<td></td>
<td>WIDGET_SLABEL</td>
<td>Widget with scroll bars for displaying text</td>
</tr>
<tr>
<td></td>
<td>WIDGET_STRING</td>
<td>Widget for entering a string</td>
</tr>
<tr>
<td></td>
<td>WIDGET_EDIT</td>
<td>Widget for editing items from a list</td>
</tr>
</tbody>
</table>

*Table 8-6: Basic ENVI Routines Grouped by Functional Area (Continued)*
<table>
<thead>
<tr>
<th>Functional Area</th>
<th>Routine Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI Widgets (cont.)</td>
<td>WIDGET_SLIST</td>
<td>Widget for creating items in a list</td>
</tr>
<tr>
<td></td>
<td>WIDGET_OUTF</td>
<td>Widget for saving a file to disk</td>
</tr>
<tr>
<td></td>
<td>WIDGET_OUTFM</td>
<td>Widget for saving a file to memory or disk</td>
</tr>
<tr>
<td></td>
<td>ENVI_INFO_WID</td>
<td>Displays text in a non-modal ‘report’ widget</td>
</tr>
<tr>
<td></td>
<td>RGB_GET_BANDS</td>
<td>ENVI’s Available Bands List dialog</td>
</tr>
<tr>
<td></td>
<td>ENVI_COLLECT_SPECTRA</td>
<td>ENVI’s Endmember Collection dialog</td>
</tr>
<tr>
<td>Image Displays</td>
<td>ENVI_DISP_QUERY</td>
<td>Returns display information</td>
</tr>
<tr>
<td></td>
<td>ENVI_GET_IMAGE</td>
<td>Returns stretched data from a display window</td>
</tr>
<tr>
<td></td>
<td>ENVI_GET_DISPLAY_NUMBERS</td>
<td>Returns all available ENVI display numbers</td>
</tr>
<tr>
<td></td>
<td>ENVI_DISPLAY_BANDS</td>
<td>Creates an ENVI Image Display Group</td>
</tr>
<tr>
<td></td>
<td>DISP_GET_LOCATION</td>
<td>Returns the position of the selected pixel</td>
</tr>
<tr>
<td></td>
<td>DISP_GOTO</td>
<td>Moves the cursor to a specified location</td>
</tr>
<tr>
<td>Vector Processing</td>
<td>ENVI_EVF_OPEN</td>
<td>Opens an EVF file and returns an EVF ID</td>
</tr>
<tr>
<td></td>
<td>ENVI_EVF_INFO</td>
<td>Returns info about an EVF file</td>
</tr>
<tr>
<td></td>
<td>ENVI_EVF_READ_RECORD</td>
<td>Reads EVF records into an IDL variable</td>
</tr>
<tr>
<td></td>
<td>ENVI_EVF_DEFINE_INIT</td>
<td>Creates a new EVF ID</td>
</tr>
<tr>
<td></td>
<td>ENVI_EVF_DEFINE_ADD_RECORD</td>
<td>Adds records to an EVF file</td>
</tr>
<tr>
<td></td>
<td>ENVI_EVF_DEFINE_CLOSE</td>
<td>Closes an EVF ID for editing</td>
</tr>
<tr>
<td></td>
<td>ENVI_EVF_CLOSE</td>
<td>Closes an EVF file</td>
</tr>
<tr>
<td></td>
<td>ENVI_WRITE_DBF_FILE</td>
<td>Writes an EVF attribute file in DBF format</td>
</tr>
</tbody>
</table>

*Table 8-6: Basic ENVI Routines Grouped by Functional Area (Continued)*
Quick Reference of ENVI Library Function Variable Codes

Many of the ENVI library routines use integer values (or codes) to define the state of variables, data, or images. These codes are often encountered in routines such as ENVI_FILE_QUERY, ENVI_SETUP_HEAD, ENVI_INIT_TILE, and ENVI_PROJ_CREATE, although their definitions are the same in all library routines. Table 8-7 provides a quick reference of the codes for many of the most commonly used variables, including the projection type code and data required for defining a new projection in batch mode.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>byte</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>short integer (2 bytes)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>long integer (4 bytes)</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>floating point</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>double precision floating point</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>complex floating point</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>unsigned short integer</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>unsigned long integer</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>64-bit signed integer</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>64-bit unsigned integer</td>
</tr>
<tr>
<td>Interleave</td>
<td>0</td>
<td>BSQ</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>BIL</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>BIP</td>
</tr>
</tbody>
</table>

Table 8-7: Code definitions for the most commonly used variables.
### Default Stretch

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no stretch</td>
</tr>
<tr>
<td>1</td>
<td>percent linear</td>
</tr>
<tr>
<td>2</td>
<td>linear range</td>
</tr>
<tr>
<td>3</td>
<td>gaussian</td>
</tr>
<tr>
<td>4</td>
<td>equalize</td>
</tr>
<tr>
<td>5</td>
<td>square foot</td>
</tr>
</tbody>
</table>

### Byte Swap

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>data file is same byte order as the current processor</td>
</tr>
<tr>
<td>1</td>
<td>data file is different byte order than the current processor</td>
</tr>
</tbody>
</table>

### Projection Units

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>meters</td>
</tr>
<tr>
<td>1</td>
<td>km</td>
</tr>
<tr>
<td>2</td>
<td>feet</td>
</tr>
<tr>
<td>3</td>
<td>yards</td>
</tr>
<tr>
<td>4</td>
<td>miles</td>
</tr>
<tr>
<td>5</td>
<td>nautical miles</td>
</tr>
<tr>
<td>6</td>
<td>degrees</td>
</tr>
<tr>
<td>7</td>
<td>minutes</td>
</tr>
<tr>
<td>8</td>
<td>seconds</td>
</tr>
<tr>
<td>9</td>
<td>radians</td>
</tr>
<tr>
<td>10</td>
<td>acres</td>
</tr>
<tr>
<td>11</td>
<td>hectares</td>
</tr>
<tr>
<td>others</td>
<td>Use ENVI_TRANSLATE_PROJECTION_UNITS</td>
</tr>
</tbody>
</table>

*Table 8-7: Code definitions for the most commonly used variables.*
<table>
<thead>
<tr>
<th>Variable</th>
<th>Type Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Arbitrary [no parameters required]</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Geographic lat/lon [no parameters required]</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>UTM [zone number]</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Transverse Mercator [a, b, lat0, lon0, x0, y0, k0]</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Lambert Conformal Conic [a, b, lat0, lon0, x0, y0, sp1, sp2]</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>Hotine Oblique Mercator A [a, b, lat0, lat1, lon1, lat2, lon2, x0, y0, k0]</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Hotine Oblique Mercator B [a, b, lat0, lon0, azimuth, x0, y0, k0]</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>Stereographic (ellipsoid) [a, b, lat0, lon0, x0, y0, k0]</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>State Plane [zone number]</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>Albers Conical Equal Area [a, b, lat0, lon0, x0, y0, sp1, sp2]</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>Polyconic [a, b, lat0, lon0, x0, y0]</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>Lambert Azimuthal Equal Area [a, b, lat0, lon0, x0, y0]</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>Azimuthal Equidistant [r, lat0, lon0, x0, y0]</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>Gnomonic [r, lat0, lon0, x0, y0]</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>Orthographic [r, lat0, lon0, x0, y0]</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>General Vertical Nearside Perspective [r, lat0, lon0, x0, y0, height]</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>Sinusoidal [r, lon0, x0, y0]</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>Equirectangular [r, lat0, lon0, x0, y0]</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>Miller Cylindrical [r, lon0, x0, y0]</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>Van der Griten [r, lat0, lon0, x0, y0]</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>Mercator [a, b, lat0, lon0, x0, y0]</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>Robinson [r, lon0, x0, y0]</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>Space Oblique Mercator A [a, b, sat num, path num, path flag, x0, y0]</td>
</tr>
<tr>
<td>Variable</td>
<td>Type Code</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>23 Alaska Conformal</td>
<td>23</td>
<td>[a, b, x0, y0]</td>
</tr>
<tr>
<td>24 Interrupted Goode</td>
<td>24</td>
<td>[r]</td>
</tr>
<tr>
<td>25 Mollweide</td>
<td>25</td>
<td>[lon0, x0, y0]</td>
</tr>
<tr>
<td>26 Interrupted Mollweide</td>
<td>26</td>
<td>[r]</td>
</tr>
<tr>
<td>27 Hammer</td>
<td>27</td>
<td>[r, lon0, x0, y0]</td>
</tr>
<tr>
<td>28 Wagner IV</td>
<td>28</td>
<td>[r, lon0, x0, y0]</td>
</tr>
<tr>
<td>29 Wagner VII</td>
<td>29</td>
<td>[r, lon0, x0, y0]</td>
</tr>
<tr>
<td>30 Oblated Equal Area</td>
<td>30</td>
<td>[r, lat0, lon0, x0, y0, shapem, shapen, angle]</td>
</tr>
<tr>
<td>31 Polar Stereographic</td>
<td>31</td>
<td>[a, b, lat0, lon0, x0, y0]</td>
</tr>
<tr>
<td>32 Space Oblique Mercator B</td>
<td>32</td>
<td>[a, b, sat num, path num, x0, y0]</td>
</tr>
<tr>
<td>33 Equidistant Conic A</td>
<td>33</td>
<td>[a, b, lat0, lon0, x0, y0, sp1]</td>
</tr>
<tr>
<td>34 Equidistant Conic B</td>
<td>34</td>
<td>[a, b, lat0, lon0, x0, y0, sp1, sp2]</td>
</tr>
<tr>
<td>35 Stereographic (sphere)</td>
<td>35</td>
<td>[r, lat0, lon0, x0, y0]</td>
</tr>
<tr>
<td>36 Lambert Azimuthal Equal Area (sphere)</td>
<td>36</td>
<td>[r, lat0, lon0, x0, y0]</td>
</tr>
<tr>
<td>37 Space Oblique Mercator A</td>
<td>37</td>
<td>[a, b, IncAng, AscLong, x0, y0, PsRev, LRat]</td>
</tr>
<tr>
<td>38 Integerized Sinusoidal</td>
<td>38</td>
<td>[r, lon0, x0, y0, Dzone, Djustify]</td>
</tr>
<tr>
<td>99 User Defined</td>
<td>99</td>
<td>[a, b, lat0, lon0, x0, y0, [additional parameters]]</td>
</tr>
</tbody>
</table>

*Table 8-7: Code definitions for the most commonly used variables.*
Chapter 9: ENVI Routines

This chapter covers the following topics:

<table>
<thead>
<tr>
<th>How to Use this Chapter</th>
<th>202</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available Routines</td>
<td>205</td>
</tr>
</tbody>
</table>
How to Use this Chapter

All of ENVI’s routines are documented alphabetically in this chapter. A description of each routine follows its name. Beneath the general description of the routine are a number of sections that describe the calling sequence for the routine, its arguments (if any), its keywords (if any), and an example of using the routine. These sections are described below.

Calling Sequence

The “Calling Sequence” section shows the proper syntax for calling the function or procedure.

Procedures

Procedures have the calling sequence:

    PROCEDURE_NAME, Argument [, Optional_Arguments]

where PROCEDURE_NAME is the name of the procedure, Argument is a required parameter, and Optional_Argument is an optional parameter to the procedure. Note that the square brackets around optional arguments are not used in the actual call to the procedure, they are simply used to denote the optional nature of the arguments within this document.

Functions

Functions have the calling sequence:

    Result = FUNCTION_NAME(Argument [, Optional_Arguments])

where Result is the returned value of the function, FUNCTION_NAME is the name of the function, Argument is a required parameter, and Optional_Argument is an optional parameter. Note that the square brackets around optional arguments are not used in the actual call to the function, they are simply used to denote the optional nature of the arguments within this document. Note also that all arguments and keyword arguments to functions should be supplied within the parentheses that follow the function’s name.

Functions do not always have to be used in assignment statements (i.e., A=SIN(10.2)), they can be used just like any other IDL expression. For example, you could print the result of SIN(10.2) by entering the command:

    PRINT, SIN(10.2)
Arguments

The “Arguments” section describes each valid argument to the routine. Note that these arguments are positional parameters that must be supplied in the order indicated by the routine’s calling sequence.

Named Variables

Often, arguments that contain values upon return from the function or procedure (“output arguments”) are described as accepting “named variables”. A named variable is simply a valid IDL variable name. This variable does not need to be defined before being used as an output argument. Note, however that when an argument calls for a named variable, only a named variable can be used—sending an expression causes an error.

Keywords

The “Keywords” section describes each valid keyword argument to the routine. Note that keyword arguments are formal parameters that can be supplied in any order.

Keyword arguments are supplied to ENVI routines by including the keyword name followed by an equal sign (“=” ) and the value to which the keyword should be set. For example, to set the filename to ENVI_SETUP_HEAD set the FNAME keyword to a string containing the desired filename.

```idl
ENVI_SETUP_HEAD, FNAME='test.img'
```

Note that keywords can be abbreviated to their shortest unique length. However, this is not recommended, since the addition of a keyword may make the abbreviation invalid.

Setting Keywords

When the documentation for a keyword says something similar to, “Set this keyword to enable logarithmic plotting,” the keyword is simply a switch that turns an option on and off. Usually, setting such keywords equal to 1 causes the option to be turned on. Explicitly setting the keyword to zero (or not including the keyword) turns the option off.

There is a “shortcut” that can be used to set a keyword equal to 1 without the usual syntax (i.e., `KEYWORD=1`). To “set” a keyword, simply preface it with a slash character (“/”). For example, set the OPEN keyword to ENVI_SETUP_HEAD as follows:

```idl
ENVI_SETUP_HEAD, FNAME=FNAME, /OPEN
```
Named Variables

Often, keywords that contain values upon return from the function or procedure (“output arguments”) are described as accepting “named variables”. A named variable is simply a valid IDL variable name. This variable does not need to be defined before being used with the keyword. Note, however that when a keyword calls for a named variable, only a named variable can be used—sending an expression causes an error.

For example, the WIDGET_CONTROL procedure can return the user values of widgets in a named variable using the GET_UVALUE keyword. To return the user value for a widget ID (contained in the variable MYWIDGET) in the variable USERVAL, you would use the command:

```
WIDGET_CONTROL, mywidget, GET_UVALUE = userval
```

Upon return from the procedure, USERVAL contains the user value. Note that USERVAL did not have to be defined before the call to WIDGET_CONTROL.

Examples

Most routines have an example that demonstrates how the function or procedure is used. These code fragments are designed to serve as examples for your own programs.

See Also

The names of related routines will be listed in this section
Available Routines

The following section lists all routines available to the ENVI developer. Table 9-1 shows the routine name, a brief description, and the reference page number.

**Note**

All of the batch processing function examples assume that the core ENVI save files have been restored. For more information on restoring the core ENVI save files see “Batch Mode ENVI” on page 53.

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAPT_FILT_DOIT</td>
<td>Perform Adaptive filtering</td>
<td>217</td>
</tr>
<tr>
<td>AIRSAR_HEADER_DOIT</td>
<td>Read AIRSAR Header</td>
<td>221</td>
</tr>
<tr>
<td>AIRSAR_PED_HEIGHT_DOIT</td>
<td>Calculate pedestal height images from AIRSAR compressed stokes matrix</td>
<td>223</td>
</tr>
<tr>
<td>AIRSAR_PHASE_IMAGE_DOIT</td>
<td>Calculate phase images from AIRSAR compressed stokes matrix</td>
<td>226</td>
</tr>
<tr>
<td>AIRSAR_POLSIG_DOIT</td>
<td>Calculate polarization signatures from AIRSAR compressed stokes matrix</td>
<td>229</td>
</tr>
<tr>
<td>AIRSAR_SCATTER_DOIT</td>
<td>Calculate scatter classification for AIRSAR compressed stokes matrix</td>
<td>232</td>
</tr>
<tr>
<td>AIRSAR_SYNTH_DOIT</td>
<td>Synthesize AIRSAR images</td>
<td>236</td>
</tr>
<tr>
<td>ASPECT_DOIT</td>
<td>Make aspect corrections to Landsat MSS image data</td>
<td>240</td>
</tr>
<tr>
<td>AUTO_WID_MNG</td>
<td>Automatically perform event handling for ENVI compound widgets</td>
<td>243</td>
</tr>
<tr>
<td>BAD_DATA_DOIT</td>
<td>Remove bad data lines</td>
<td>244</td>
</tr>
</tbody>
</table>

Table 9-1: ENVI Routines
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATCH</td>
<td>Establish an error handler for the current procedure</td>
<td>247</td>
</tr>
<tr>
<td>CF_DOIT</td>
<td>Create an output file</td>
<td>248</td>
</tr>
<tr>
<td>CLASS_CONFUSION_DOIT</td>
<td>Compute classification confusion matrix</td>
<td>251</td>
</tr>
<tr>
<td>CLASS_CS_DOIT</td>
<td>CLUMP and SIEVE a classification image</td>
<td>256</td>
</tr>
<tr>
<td>CLASS_DOIT</td>
<td>Perform supervised classification</td>
<td>260</td>
</tr>
<tr>
<td>CLASS MAJORITY_DOIT</td>
<td>Perform majority or minority analysis on a classification image</td>
<td>270</td>
</tr>
<tr>
<td>CLASS_RULE_DOIT</td>
<td>Classify Rule Images</td>
<td>273</td>
</tr>
<tr>
<td>CLASS_STATS_DOIT</td>
<td>Calculate class statistics</td>
<td>276</td>
</tr>
<tr>
<td>COM_CLASS_DOIT</td>
<td>Combine Classes</td>
<td>280</td>
</tr>
<tr>
<td>CONTINUUM_REMOVE_DOIT</td>
<td>Perform continuum removal</td>
<td>283</td>
</tr>
<tr>
<td>CONV_DOIT</td>
<td>Perform convolution filtering</td>
<td>286</td>
</tr>
<tr>
<td>CONVERT_DOIT</td>
<td>Convert interleave type (BSQ, BIL, BIP)</td>
<td>289</td>
</tr>
<tr>
<td>CONVERT_INPLACE_DOIT</td>
<td>Convert in place between storage types</td>
<td>291</td>
</tr>
<tr>
<td>CROSS_TRACK_CORRECTION_DOIT</td>
<td>Remove variation in the cross-track illumination of an image</td>
<td>293</td>
</tr>
<tr>
<td>DARK_SUB_DOIT</td>
<td>Perform Dark Subtraction</td>
<td>297</td>
</tr>
<tr>
<td>DECOR_DOIT</td>
<td>Perform Saturation Stretch</td>
<td>300</td>
</tr>
<tr>
<td>DEM_BAD_DATA_DOIT</td>
<td>Correct bad data points in DEMs</td>
<td>303</td>
</tr>
<tr>
<td>DESKEW_DOIT</td>
<td>Deskew MSS</td>
<td>306</td>
</tr>
<tr>
<td>DESTRIPE_DOIT</td>
<td>Destripe image data</td>
<td>309</td>
</tr>
</tbody>
</table>

Table 9-1: ENVI Routines (Continued)
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP_GET_LOCATION</td>
<td>Get x and y pixel locations in a display</td>
<td>312</td>
</tr>
<tr>
<td>DISP_GOTO</td>
<td>Move the current pixel in a display</td>
<td>314</td>
</tr>
<tr>
<td>DISP_OUT_IMG</td>
<td>Output to Postscript</td>
<td>317</td>
</tr>
<tr>
<td>ELINE_CAL_DOIT</td>
<td>Perform Empirical Line Calibration</td>
<td>320</td>
</tr>
<tr>
<td>EMITTANCE_CALC_DOIT</td>
<td>Convert emissivity</td>
<td>323</td>
</tr>
<tr>
<td>ENVI</td>
<td>Restore base ENVI Save files for Batch mode.</td>
<td>326</td>
</tr>
<tr>
<td>ENVI_ADD_PROJECTION</td>
<td>Add a projection to ENVI</td>
<td>327</td>
</tr>
<tr>
<td>ENVI_AVHRR_CALIBRATE_DOIT</td>
<td>Calibrate AVHRR data or compute AVHRR Sea Surface Temperature (SST)</td>
<td>328</td>
</tr>
<tr>
<td>ENVI_AVHRR_GEOMETRY_DOIT</td>
<td>Compute the AVHRR geometry (latitude and longitude), solar zenith angles, and sensor zenith angles for each pixel</td>
<td>331</td>
</tr>
<tr>
<td>ENVI_AVHRR_WARP_DOIT</td>
<td>Warp AVHRR data or resulting AVHRR data products</td>
<td>334</td>
</tr>
<tr>
<td>ENVI_BATCH_EXIT</td>
<td>Exit ENVI from the non-menu “batch” mode</td>
<td>339</td>
</tr>
<tr>
<td>ENVI_BATCH_INIT</td>
<td>Initialize ENVI in the non-menu “batch” mode</td>
<td>341</td>
</tr>
<tr>
<td>ENVI_BATCH_STATUS_WINDOW</td>
<td>Enable and disable the ENVI batch status window</td>
<td>342</td>
</tr>
<tr>
<td>ENVI_BUFFER_ZONE_DOIT</td>
<td>Create a buffer zone image from a classification image</td>
<td>345</td>
</tr>
<tr>
<td>ENVI_CAL_DOIT</td>
<td>Spectrally calibrate images using FLat Field or IARR</td>
<td>348</td>
</tr>
</tbody>
</table>

Table 9-1: ENVI Routines (Continued)
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_CENTER</td>
<td>Return the centering offsets for a widget</td>
<td>351</td>
</tr>
<tr>
<td>ENVI_CLOVER_DOIT</td>
<td>Overlay Classes</td>
<td>352</td>
</tr>
<tr>
<td>ENVI_COLLECT_SPECTRA</td>
<td>Perform endmember collection</td>
<td>355</td>
</tr>
<tr>
<td>ENVI_COMPUTE_SUNANGLES</td>
<td>Compute sun angles</td>
<td>358</td>
</tr>
<tr>
<td>ENVI_CONVERT_FILE_COORDINATES</td>
<td>Convert between map and pixel coordinates</td>
<td>360</td>
</tr>
<tr>
<td>ENVI_CONVERT_FILE_MAP_PROJECTION</td>
<td>Convert a file from its current map projection to specified output projection</td>
<td>362</td>
</tr>
<tr>
<td>ENVI_CONVERT_PROJECTION_COORDINATES</td>
<td>Convert map coordinates between projections</td>
<td>366</td>
</tr>
<tr>
<td>ENVI_CREATE_ROI</td>
<td>Create a new ROI</td>
<td>369</td>
</tr>
<tr>
<td>ENVI_CUBE_3D_DOIT</td>
<td>Build a 3D Image Cube</td>
<td>370</td>
</tr>
<tr>
<td>ENVIDEFINE_ROI</td>
<td>Add objects to an ROI</td>
<td>376</td>
</tr>
<tr>
<td>ENVI_DEFAULT_STRETCH_CREATE</td>
<td>Return an ENVI default stretch structure</td>
<td>373</td>
</tr>
<tr>
<td>ENVI_DELETE_ROIS</td>
<td>Delete ROIs</td>
<td>379</td>
</tr>
<tr>
<td>ENVI_DISP_QUERY</td>
<td>Get display information</td>
<td>380</td>
</tr>
<tr>
<td>ENVI_DISPLAY_BANDS</td>
<td>Display grayscale or RGB image in a display window</td>
<td>383</td>
</tr>
<tr>
<td>ENVI_DOIT</td>
<td>Interface for all of the ENVI processing routines (the _DOIT routines)</td>
<td>386</td>
</tr>
<tr>
<td>ENVI_ENTER_DATA</td>
<td>Enter image data into ENVI</td>
<td>389</td>
</tr>
<tr>
<td>ENVI_EVF_CLOSE</td>
<td>Close an ENVI Vector File (EVF)</td>
<td>396</td>
</tr>
<tr>
<td>ENVI_EVF_DEFINE_ADD_RECORD</td>
<td>Add a record to a new ENVI Vector File (EVF)</td>
<td>398</td>
</tr>
</tbody>
</table>

*Table 9-1: ENVI Routines (Continued)*
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_EVF_DEFINE_CLOSE</td>
<td>Close a new ENVI Vector File (EVF)</td>
<td>400</td>
</tr>
<tr>
<td>ENVI_EVF_DEFINE_INIT</td>
<td>Initialize a new ENVI Vector File (EVF)</td>
<td>402</td>
</tr>
<tr>
<td>ENVI_EVF_INFO</td>
<td>Get general information about an ENVI Vector File (EVF)</td>
<td>404</td>
</tr>
<tr>
<td>ENVI_EVF_OPEN</td>
<td>Open an existing ENVI Vector File (EVF)</td>
<td>406</td>
</tr>
<tr>
<td>ENVI_EVF_READ_RECORD</td>
<td>Retrieve records from an ENVI Vector File (EVF)</td>
<td>408</td>
</tr>
<tr>
<td>ENVI_FILE_MNG</td>
<td>Remove and delete ENVI files</td>
<td>410</td>
</tr>
<tr>
<td>ENVI_FILE_QUERY</td>
<td>Get information on ENVI files</td>
<td>411</td>
</tr>
<tr>
<td>ENVI_FILE_TYPE</td>
<td>File type description</td>
<td>417</td>
</tr>
<tr>
<td>ENVI_FILTER_DOIT</td>
<td>FFT Filter Building</td>
<td>420</td>
</tr>
<tr>
<td>ENVI_GET_CONFIGURATION_VALUES</td>
<td>Get current setting for ENVI configuration items</td>
<td>422</td>
</tr>
<tr>
<td>ENVI_GET_DATA</td>
<td>Get spatial image data from a file</td>
<td>423</td>
</tr>
<tr>
<td>ENVI_GET_DISPLAY_NUMBERS</td>
<td>Get a list of display numbers</td>
<td>426</td>
</tr>
<tr>
<td>ENVI_GET_FILE_IDS</td>
<td>Get file ids for all open files</td>
<td>427</td>
</tr>
<tr>
<td>ENVI_GET_IMAGE</td>
<td>Get spatial image data from a display</td>
<td>428</td>
</tr>
<tr>
<td>ENVI_GET_MAP_INFO</td>
<td>Get map information from a display</td>
<td>430</td>
</tr>
<tr>
<td>ENVI_GET_PROJECTION</td>
<td>Get the projection information for the specified file, display or map information handle</td>
<td>431</td>
</tr>
<tr>
<td>ENVI_GET_RGB_TRIPLETS</td>
<td>Get RGB triplets for graphics color index</td>
<td>433</td>
</tr>
<tr>
<td>ENVI_GET_ROI</td>
<td>Get ROI addresses</td>
<td>435</td>
</tr>
</tbody>
</table>

Table 9-1: ENVI Routines (Continued)
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_GET_ROI_DATA</td>
<td>Get the data associated with an ROI</td>
<td>436</td>
</tr>
<tr>
<td>ENVI_GET_ROI_DIMS_PTR</td>
<td>Get DIMS pointer from ROI IDs</td>
<td>438</td>
</tr>
<tr>
<td>ENVI_GET_ROI_IDS</td>
<td>Get ROI IDs</td>
<td>439</td>
</tr>
<tr>
<td>ENVI_GET_SLICE</td>
<td>Get spectral slice of data from a file</td>
<td>441</td>
</tr>
<tr>
<td>ENVI_GET_STATISTICS</td>
<td>Get values from an ENVI statistics file</td>
<td>443</td>
</tr>
<tr>
<td>ENVI_GET_TILE</td>
<td>Get tile data</td>
<td>446</td>
</tr>
<tr>
<td>ENVI_GRID_DOIT</td>
<td>Convert irregularly gridded points to raster image</td>
<td>448</td>
</tr>
<tr>
<td>ENVI_INFO_WID</td>
<td>Display text data in widget</td>
<td>451</td>
</tr>
<tr>
<td>ENVI_INIT_TILE</td>
<td>Initialize the tile processing</td>
<td>452</td>
</tr>
<tr>
<td>ENVI_IO_ERROR</td>
<td>Report Input/Output processing errors</td>
<td>454</td>
</tr>
<tr>
<td>ENVI_LAYER_STACKING_DOIT</td>
<td>Build a new multiband file from georeferenced images of various pixel sizes, extents, and projections</td>
<td>455</td>
</tr>
<tr>
<td>ENVI_MASK_APPLY_DOIT</td>
<td>Apply a mask to a file</td>
<td>465</td>
</tr>
<tr>
<td>ENVI_MAP_INFO_CREATE</td>
<td>Create map projections</td>
<td>460</td>
</tr>
<tr>
<td>ENVI_NEURAL_NET_DOIT</td>
<td>Perform classification using a Neural Net method</td>
<td>468</td>
</tr>
<tr>
<td>ENVI_OPEN_DATA_FILE</td>
<td>Open a data file</td>
<td>473</td>
</tr>
<tr>
<td>ENVI_OPEN_FILE</td>
<td>Open an ENVI file</td>
<td>476</td>
</tr>
<tr>
<td>ENVI_OUTPUT_TO_EXTERNAL_FORMAT</td>
<td>Output image data to external formats</td>
<td>477</td>
</tr>
<tr>
<td>ENVI_PICKFILE</td>
<td>Pick a filename</td>
<td>479</td>
</tr>
<tr>
<td>ENVI_PLOT_DATA</td>
<td>Create an x,y plot</td>
<td>481</td>
</tr>
<tr>
<td>ENVI_PROJ_CREATE</td>
<td>Create an ENVI projection structure</td>
<td>483</td>
</tr>
</tbody>
</table>

Table 9-1: ENVI Routines (Continued)
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_READ_COLS</td>
<td>Read ASCII column data</td>
<td>487</td>
</tr>
<tr>
<td>ENVI_REPORT_INC</td>
<td>Set the status report increment</td>
<td>488</td>
</tr>
<tr>
<td>ENVI_REPORT_INIT</td>
<td>Initialize the status report</td>
<td>489</td>
</tr>
<tr>
<td>ENVI_REPORT_STAT</td>
<td>Update status report percent complete</td>
<td>491</td>
</tr>
<tr>
<td>ENVI_RESTORE_ROIS</td>
<td>Restore saved ROI files</td>
<td>493</td>
</tr>
<tr>
<td>ENVI_ROI_TO_IMAGE_DOIT</td>
<td>Create a classification image from ROIs</td>
<td>494</td>
</tr>
<tr>
<td>ENVI_SAVE_ROIS</td>
<td>Save ROIs</td>
<td>496</td>
</tr>
<tr>
<td>ENVI_SEGMENT_DOIT</td>
<td>Segment a classification image into unique spatially contiguous “blobs”</td>
<td>497</td>
</tr>
<tr>
<td>ENVI_SELECT</td>
<td>Select an open ENVI file or band</td>
<td>500</td>
</tr>
<tr>
<td>ENVI_SENSOR_TYPE</td>
<td>Return sensor type description</td>
<td>504</td>
</tr>
<tr>
<td>ENVI_SET_INHERITANCE</td>
<td>Return the ENVI inheritance structure</td>
<td>507</td>
</tr>
<tr>
<td>ENVI_SETUP_HEAD</td>
<td>Set and write ENVI header information</td>
<td>510</td>
</tr>
<tr>
<td>ENVI_STATS_DOIT</td>
<td>Calculate statistics of a data file</td>
<td>519</td>
</tr>
<tr>
<td>ENVI_SUM_DATA_DOIT</td>
<td>Calculate a number of statistical parameters on a set of bands</td>
<td>525</td>
</tr>
<tr>
<td>ENVI_SYNTHETIC_COLOR_DOIT</td>
<td>Calculate synthetic color image</td>
<td>529</td>
</tr>
<tr>
<td>ENVI_TILE_DONE</td>
<td>Complete tile processing</td>
<td>532</td>
</tr>
<tr>
<td>ENVI_TOGGLE_CATCH</td>
<td>Toggle the ENVI error catching handling mechanism on and off</td>
<td>533</td>
</tr>
<tr>
<td>ENVI_TRANSLATE_PROJECTION_NAME</td>
<td>Convert projection names</td>
<td>534</td>
</tr>
<tr>
<td>ENVI_TRANSLATE_PROJECTION_UNITS</td>
<td>Translate projection units</td>
<td>535</td>
</tr>
</tbody>
</table>

*Table 9-1: ENVI Routines (Continued)*
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI_USER_DEFINED_ANNOTATION</td>
<td>Create an ENVI annotation file or append items to an existing annotation file</td>
<td>536</td>
</tr>
<tr>
<td>ENVI_WRITE_DBF_FILE</td>
<td>Write a DBF attribute file for an ENVI Vector File (EVF)</td>
<td>543</td>
</tr>
<tr>
<td>ENVI_WRITE_STATISTICS</td>
<td>Write an ENVI statistics file</td>
<td>545</td>
</tr>
<tr>
<td>FFT_DOIT</td>
<td>Perform Fast Fourier Transform (FFT) filtering</td>
<td>548</td>
</tr>
<tr>
<td>FFT_INV_DOIT</td>
<td>Apply FFT filter and perform inverse FFT</td>
<td>551</td>
</tr>
<tr>
<td>GAINOFF_DOIT</td>
<td>Apply Gain and Offset to each input band</td>
<td>554</td>
</tr>
<tr>
<td>GEN_IMAGE_DOIT</td>
<td>Generate test images</td>
<td>557</td>
</tr>
<tr>
<td>HANDLE_VALUE</td>
<td>Get and set handle values</td>
<td>560</td>
</tr>
<tr>
<td>HIST_EXPORT_DOIT</td>
<td>Output images using applied Lookup Table (LUT)</td>
<td>562</td>
</tr>
<tr>
<td>MAGIC_MEM_CHECK</td>
<td>Clear out the necessary memory cache for in-memory functions</td>
<td>566</td>
</tr>
<tr>
<td>MATCH_FILTER_DOIT</td>
<td>Perform Matched Filtering</td>
<td>568</td>
</tr>
<tr>
<td>MATCH_FILTER_MT_DOIT</td>
<td>Perform Mixture-Tuned Matched Filtering (MTMF)</td>
<td>572</td>
</tr>
<tr>
<td>MATH_DOIT</td>
<td>Perform Band Math</td>
<td>576</td>
</tr>
<tr>
<td>MNF_DOIT</td>
<td>Perform Minimum Noise Fraction (MNF) transform</td>
<td>579</td>
</tr>
<tr>
<td>MNF_INV_DOIT</td>
<td>Perform inverse MNF transform</td>
<td>583</td>
</tr>
<tr>
<td>MORPH_DOIT</td>
<td>Perform morphological filtering</td>
<td>585</td>
</tr>
<tr>
<td>MOSAIC_DOIT</td>
<td>Mosaic image bands or band combinations</td>
<td>588</td>
</tr>
</tbody>
</table>

Table 9-1: ENVI Routines (Continued)
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSSCAL_DOIT</td>
<td>Calibrate MSS data to radiance or reflectance</td>
<td>594</td>
</tr>
<tr>
<td>MUNSELL_DOIT</td>
<td>Convert RGB image to Munsell coordinates</td>
<td>598</td>
</tr>
<tr>
<td>MUNSELL_INV_DOIT</td>
<td>Calculate RGB image from Munsell coordinates</td>
<td>600</td>
</tr>
<tr>
<td>NDVI_DOIT</td>
<td>Create Normalized Difference Vegetation Index (NDVI)</td>
<td>602</td>
</tr>
<tr>
<td>ON_IOERROR</td>
<td>Specifies a statement to be jumped to if an Input/Output error occurs in the current procedure</td>
<td>605</td>
</tr>
<tr>
<td>PC_ROTATE</td>
<td>Calculate Principal Components Rotation</td>
<td>606</td>
</tr>
<tr>
<td>PPI_DOIT</td>
<td>Calculate Pixel Purity Index (PPI)</td>
<td>610</td>
</tr>
<tr>
<td>RADAR_INC_ANGLE_DOIT</td>
<td>Calculate incidence angle image from radar</td>
<td>613</td>
</tr>
<tr>
<td>RATIO_DOIT</td>
<td>Calculate Band Ratios</td>
<td>616</td>
</tr>
<tr>
<td>REG_WARP_DOIT</td>
<td>Warp image data</td>
<td>619</td>
</tr>
<tr>
<td>RESIZE_DOIT</td>
<td>Spatially resize data</td>
<td>624</td>
</tr>
<tr>
<td>RGB_GET_BANDS</td>
<td>Display a dialog for selecting three bands from the Available Bands List</td>
<td>627</td>
</tr>
<tr>
<td>RGB_ITRANS_DOIT</td>
<td>Perform inverse color transform</td>
<td>629</td>
</tr>
<tr>
<td>RGB_TRANS_DOIT</td>
<td>Perform forward color transform</td>
<td>632</td>
</tr>
<tr>
<td>ROC_CURVE_DOIT</td>
<td>Compute Receiver Operating Characteristic (ROC) curves</td>
<td>635</td>
</tr>
<tr>
<td>ROI_THRESH_DOIT</td>
<td>Threshold ROI (Region of Interest)</td>
<td>638</td>
</tr>
<tr>
<td>ROTATE_DOIT</td>
<td>Rotate images</td>
<td>641</td>
</tr>
<tr>
<td>RTV_DOIT</td>
<td>Perform raster to vector conversion</td>
<td>644</td>
</tr>
</tbody>
</table>

Table 9-1: ENVI Routines (Continued)
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT_STRETCH_DOIT</td>
<td>Perform Saturation Stretch</td>
<td>647</td>
</tr>
<tr>
<td>SHARPEN_DOIT</td>
<td>Perform Image Sharpening</td>
<td>650</td>
</tr>
<tr>
<td>SIRC_HEADER_DOIT</td>
<td>Read SIR-C Header</td>
<td>654</td>
</tr>
<tr>
<td>SIRC_MULTILOOK_DOIT</td>
<td>Multilook SIR-C Images</td>
<td>657</td>
</tr>
<tr>
<td>SIRC_PED_HEIGHT_DOIT</td>
<td>Calculate pedestal height images from a SIR-C compressed data products file</td>
<td>661</td>
</tr>
<tr>
<td>SIRC_PHASE_IMAGE_DOIT</td>
<td>Calculate phase images from a SIR-C compressed data products file</td>
<td>665</td>
</tr>
<tr>
<td>SIRC_POLSIG_DOIT</td>
<td>Calculate polarization signatures from a SIR-C compressed data products file</td>
<td>669</td>
</tr>
<tr>
<td>SIRC_SYNTH_DOIT</td>
<td>Synthesize SIR-C Images</td>
<td>673</td>
</tr>
<tr>
<td>SLICE_DOIT</td>
<td>Extract a spectral slice</td>
<td>678</td>
</tr>
<tr>
<td>SLT2GND_DOIT</td>
<td>Convert slant to ground range</td>
<td>680</td>
</tr>
<tr>
<td>SPECTRAL_FEATURE_DOIT</td>
<td>Perform Spectral Feature Fitting</td>
<td>683</td>
</tr>
<tr>
<td>STRETCH_DOIT</td>
<td>Perform Contrast Stretch of image data</td>
<td>687</td>
</tr>
<tr>
<td>TASCAP_DOIT</td>
<td>Create Tasseled Cap vegetation and soil indices</td>
<td>691</td>
</tr>
<tr>
<td>TEXTURE_COOCCUR_DOIT</td>
<td>Calculate Co-occurrence Texture Measures</td>
<td>694</td>
</tr>
<tr>
<td>TEXTURE_STATS_DOIT</td>
<td>Calculate Occurrence Texture Measures</td>
<td>698</td>
</tr>
<tr>
<td>TIMS_CAL_DOIT</td>
<td>Calibrate TIMS data to radiance</td>
<td>701</td>
</tr>
<tr>
<td>TMCAL_DOIT</td>
<td>Calibrate Landsat TM data to radiance or reflectance</td>
<td>704</td>
</tr>
</tbody>
</table>

Table 9-1: ENVI Routines (Continued)
### Table 9-1: ENVI Routines (Continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOPO_DOIT</td>
<td>Perform Topographic Modeling of DEMs</td>
<td>708</td>
</tr>
<tr>
<td>TOPO_FEATURE_DOIT</td>
<td>Create a topographic feature classification image</td>
<td>712</td>
</tr>
<tr>
<td>UNMIX_DOIT</td>
<td>Perform Linear Spectral Unmixing</td>
<td>717</td>
</tr>
<tr>
<td>VAX_IEEE_DOIT</td>
<td>Perform VAX IEEE Floating Point Conversion</td>
<td>720</td>
</tr>
<tr>
<td>WIDGET_AUTO_BASE</td>
<td>Create top level base for auto-managed widgets</td>
<td>722</td>
</tr>
<tr>
<td>WIDGET_BASE</td>
<td>Create a base widget</td>
<td>724</td>
</tr>
<tr>
<td>WIDGET_CONTROL</td>
<td>Realize, manage, and destroy widgets</td>
<td>729</td>
</tr>
<tr>
<td>WIDGET_EDIT</td>
<td>Return the base ID of the edit list widget</td>
<td>735</td>
</tr>
<tr>
<td>WIDGET_GEO</td>
<td>Return the base ID of the latitude/longitude specification widget</td>
<td>738</td>
</tr>
<tr>
<td>WIDGET_MAP</td>
<td>Return the base ID of the map information input/edit widget</td>
<td>740</td>
</tr>
<tr>
<td>WIDGET_MENU</td>
<td>Return the base ID of the menu widget</td>
<td>742</td>
</tr>
<tr>
<td>WIDGET_MULTI</td>
<td>Return the base ID of the multiple list selection widget</td>
<td>744</td>
</tr>
<tr>
<td>WIDGET_OUTF</td>
<td>Return the base ID of the output filename selection widget</td>
<td>746</td>
</tr>
<tr>
<td>WIDGET_OUTFM</td>
<td>Return the base ID of the output filename or memory selection widget</td>
<td>748</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>WIDGET_PARAM</td>
<td>Return the base ID of the parameter input widget</td>
<td>750</td>
</tr>
<tr>
<td>WIDGET_PMENU</td>
<td>Return the base ID of the pulldown menu widget</td>
<td>753</td>
</tr>
<tr>
<td>WIDGET_RGB</td>
<td>Return the base ID of the color value editing widget</td>
<td>755</td>
</tr>
<tr>
<td>WIDGET_SLABEL</td>
<td>Return the base ID of the display string widget</td>
<td>757</td>
</tr>
<tr>
<td>WIDGET_SLIST</td>
<td>Return the base ID of the list widget</td>
<td>758</td>
</tr>
<tr>
<td>WIDGET_SSLIDER</td>
<td>Return the base ID of the slider widget</td>
<td>760</td>
</tr>
<tr>
<td>WIDGET_STRING</td>
<td>Return the base ID of the string entry widget</td>
<td>763</td>
</tr>
<tr>
<td>WIDGET_SUBSET</td>
<td>Return the base ID of the image subset widget</td>
<td>765</td>
</tr>
<tr>
<td>WIDGET_TOGGLE</td>
<td>Return the base ID of the arrow toggle selection widget</td>
<td>767</td>
</tr>
</tbody>
</table>

*Table 9-1: ENVI Routines (Continued)*
ADAPT_FILT_DOIT

Use this program to perform adaptive filtering including Lee filter, localized sigma filter and bit error removal.

Calling Sequence

ENVI_DOIT, 'ADAPT_FILT_DOIT'

Keywords

ADD_MEAN

Use this keyword to specify a variable that will contain the additive noise mean. ADD_MEAN is a floating point number. This value is only used for the LEE filter, METHOD 0, and NOISE_TYPE 2.

DAMP

Use this keyword to specify the filter damping factor. DAMP is a floating point number greater than or equal to zero. If DAMP is set to zero the filter performs like an averaging filter. This value is only used for the Frost filter, METHOD 3.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.
**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**KX**

Use this keyword to specify a square kernel size. KX is an integer number greater than or equal to three.

**METHOD**

Use this keyword to specify the type of filter. Choose one of the following:

- 0 - LEE
- 1 - Bit Error Removal
- 2 - Localized Sigma
- 3 - Frost
- 4 - Gamma
- 5 - Kuan

**MULT_MEAN**

Use this keyword to specify a variable that will contain the multiplicative noise mean. MULT_MEAN is a floating point number. This value is only used for the LEE filter, METHOD 0 and NOISE_TYPE 1 and 2.

**NLOOK**

Use this keyword to specify a variable that will contain the number of looks for the data. NLOOK is an integer number. This value is only used for the GAMMA and KUAN filters, METHOD 4 and 5 respectively.

**NOISE_TYPE**

Use this keyword to specify a variable that will contain the type of noise to filter. The two types of noise are multiplicative noise and additive noise. NOISE_TYPE is an integer value ranging from 0 to 2.

- 0 - Additive noise
- 1 - Multiplicative noise
- 2 - Both additive and multiplicative
OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

REPLACE
Set this keyword to 1 to replace bit errors with the local average. Otherwise, set this keyword to zero. This keyword is only used for method 1.

SIGMA
Use this keyword to specify Sigma for method 0 or the Sigma Factor for methods 1 and 2.

TOL
Use this keyword to specify the maximum standard deviation tolerance. This keyword is only used for method 1.

VMAX
Use this keyword to specify the maximum value for valid data. This keyword is only used for method 1.

VMIN
Use this keyword to specify the minimum value for valid data. This keyword is only used for method 1.
Example

```pro
pro example_adapt_filt_doit

; First restore all the base save files.
envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
envi_batch_init, log_file='batch.txt'

; Open the input file
envi_open_file, 'bonnrsat.img', r_fid=fid
if (fid eq -1) then begin
  envi_batch_exit
  return
endif

; Set the necessary variables
envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
kx = 5
sigma = .5
method = 0
mult_mean = 1.
noise_type = 1
out_name = 'testimg'

; Call the "doit"
envi_doit, 'adapt_filt_doit', fid=fid, pos=pos, $
  dims=dims, out_name=out_name, kx=kx, sigma=sigma, $
  noise_type=noise_type, mult_mean=mult_mean, $
  method=method, in_memory=0

; Exit ENVI
envi_batch_exit
```


AIRSAR_HEADER_DOIT

Use this function to read the AIRSAR header information used by other AIRSAR routines. The function returns one if the specified file is an AIRSAR file. Otherwise, zero is returned. The desired AIRSAR information is returned using optional keywords.

Calling Sequence

ENVI_DOIT, ‘AIRSAR_HEADER_DOIT’

Keywords

BAND (optional)

Use this keyword to specify a named variable that will contain the band number for the file specified by Filename. BAND is set to 0, 1, or 2 to specify C, L, or P respectively.

FILENAME

Set this keyword to specify the filename of the AIRSAR file.

GENFAC (optional)

Use this optional keyword to specify a named variable that will contain the COMP SCALE FACTOR for the band.

NAME

Use this keyword to specify an AIRSAR file name.

NL (optional)

Use this optional keyword to specify a named variable that will contain the number of lines in the AIRSAR image.

NS (optional)

Use this optional keyword to specify a named variable that will contain the number of samples per line in the AIRSAR image.

OFFSET (optional)

Use this optional keyword to specify a named variable that will contain the offset for the file specified by Filename.
Example

pro example_airssar_header_doit

; First restore all the base save files.
; envi, /restore_base_save_files
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; Set the keywords
; filename = 'indvc.dat'
; Call the "doit"
; envi_doit, 'airsar_header_doit', $
   ; filename=filename, ns=ns, nl=nl, $
   ; offset=offset, genfac=genfac, $
   ; band=band
; Print the result
; print, 'ns = ', ns
print, 'nl = ', nl
print, 'offset = ', offset
print, 'genfac = ', genfac
print, 'band = ', band
; Exit ENVI
; envi_batch_exit
end
AIRSAR_PED_HEIGHT_DOIT

Use this program to calculate pedestal height images from an AIRSAR compressed stokes matrix.

Calling Sequence

ENVI_DOIT, ‘AIRSAR_PED_HEIGHT_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FNAME

Use this keyword to specify a string array of compressed stokes matrix file names for C, L and/or P wavelengths respectively. If a file is not used, set the array element to ‘’. 

FNS

Use this keyword to specify the number of samples per line in the AIRSAR image.

FNL

Use this keyword to specify the number of lines in the AIRSAR image.

GENFAC

Use this keyword to specify an array of COMP SCALE FACTORS for each of the files specified by FNAME.
IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

OFFSET

Use this keyword to specify a long array of header offsets for each of the files specified by FNAME.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_airsar_ped_height_doit
   ; First restore all the base save files.
   ;
   envi, /restore_base_save_files
   ;
   ; Initialize ENVI and send all errors and warnings to the file batch.txt
   ;
   envi_batch_init, log_file='batch.txt'
   ;
   ; Set the keywords
   ;
   fname = ['indvc.dat', $
       'indvl.dat', $
       'indvp.dat']
   fns = 1024
   fnl = 1224
   out_type = 0
   out_name = 'testimg'
   offset = [30720, 30720, 30720]
   genfac = [18.1776, 6.51427, 1.63261]
```
dims = [1, 0, fns-1, 0, fnl-1]
;
; Call the "doit"
;
envi_doit, 'airsar_ped_height_doit', $
  fns=fname, offset=offset, $
  fns=fns, fnl=fnl, dims=dims, $
  out_name=out_name, genfac=genfac, $
  r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
AIRSAR_PHASE_IMAGE_DOIT

Use this program to calculate phase images from an AIRSAR compressed stokes matrix.

Calling Sequence

ENVI_DOIT, 'AIRSAR_PHASE_IMAGE_DOIT'

Keywords

DEGREES

Set this optional keyword to output the phase images in degrees. Set this keyword to zero to output the phase image in radians.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FNAME

Use this keyword to specify a string array of compressed stokes matrix file names for C, L or P wavelengths. A phase image will be calculated for each element in FNAME.

FNS

Use this keyword to specify the number of samples per line in the AIRSAR image.

FNL

Use this keyword to specify the number of lines in the AIRSAR image.
GENFAC

Use this keyword to specify an array of COMP SCALE FACTORS for each of the files specified by FNAME.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

OFFSET

Use this keyword to specify a long array of header offsets for each of the files specified by FNAME.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_airpsar_phase_image_doit
;
; First restore all the base save files.
;
envi, /restore_base_save_files
;
; Initialize ENVI and send all errors and warnings to the file batch.txt
;
envi_batch_init, log_file='batch.txt'
;
; Set the input and output file names
;
out_name = 'testimg'
fname = ['indvc.dat', $ 'indvl.dat', $ 'indvp.dat']
```
; Set the keywords
;
fs = 1024
fnl = 1224
dims = [-1, 0, fs-1, 0, fnl-1]
out_type = 0
offset = [30720, 30720, 30720]
genfac = [18.1776, 6.51427, 1.63261]
out_bname = ['Phase:indvc.dat', 
             'Phase:indvl.dat', 
             'Phase:indvp.dat']

; Call the "doit"
;
envi_doit, 'airsar_phase_image_doit', fname=fname, 
    offset=offset, dims=dims, out_name=out_name, 
    out_bname=out_bname, /degrees, genfac=genfac,
    in_memory=0, fs=fs, fnl=fnl, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
AIRSAR_POLSIG_DOIT

Use this program to calculate polarization signatures from an AIRSAR compressed stokes matrix.

Calling Sequence

ENVI_DOIT, ‘AIRSAR_POLSIG_DOIT’

Keywords

BANDS

Use this keyword to specify a three-element array of ones and zeros, indicating whether the C, L and P wavelengths were used. A value of one indicates the wavelength was used. BANDS must be three elements long, regardless of the size of FNAME.

BFNAME

Use this keyword to specify a three-element string array, where each element specifies C, L and P annotations for the header description. BFNAME must be three elements long regardless of the size of FNAME.

FNAME

Use this keyword to specify a string array of compressed stokes matrix file names for C, L and/or P wavelengths, respectively. If a file is not used, set the array element to ‘’.

FNS

Use this keyword to specify the number of samples per line in the AIRSAR image.

FNL

Use this keyword to specify the number of lines in the AIRSAR image.

GENFAC

Use this keyword to specify an array of COMP SCALE FACTORS for each of the files specified by FNAME.
IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

OFFSET

Use this keyword to specify a long array of header offsets for each of the files specified by FNAME.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

ROI_ID

Use this keyword to specify an array of ROI IDs returned from a call to ENVI_GET_ROI_IDS. Each ID in the array will use the corresponding ROI to calculate both a co-polarization and cross polarization image.

Example

```
forward_function envi_get_roi_ids

pro example_airsar_polsig_doit
;
; First restore all the base save files.
; envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
;
; Set the keywords
;
fname = ['indvc.dat', $
```
`indv.dat', 'indvp.dat']
fns = 1024
fnl = 1224
out_type = 0
out_name = 'testimg'
offset = [30720, 30720, 30720]
genfac = [18.1776, 6.51427, 1.63261]

; Restore the presaved polsig ROI
;
envi_restore_rois, 'indv.roi'
roi_id = envi_get_roi_ids(ns=fns, nl=fnl)
roi_id = [roi_id, roi_id, roi_id]

; Call the "doit"
;
envi_doit, 'airsar_polsig_doit', fname=fname, 
  offset=offset, roi_id=roi_id, 
  out_name=out_name, out_bname=out_bname, 
  genfac=genfac, in_memory=0, fns=fns, 
  fnl=fnl, bands=[1,1,1], bname=fname, 
  out_type=out_type, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
AIRSAR_SCATTER_DOIT

Use this program to classify an AIRSAR image based on the scattering.

Calling Sequence

ENVI_DOIT, ‘AIRSAR_SCATTER_DOIT’

Keywords

CLASS_NAMES

Use this keyword to specify names for each output class. CLASS_NAMES is an array of strings with num_classes+1 elements. Remember to set the zero class to “Unclassified”.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FNAME

Use this keyword to specify a string array of compressed stokes matrix file names for C, L and P wavelengths, respectively. All wavelengths must be specified for this function.

FNS

Use this keyword to specify the number of samples per line in the AIRSAR image.

FNL

Use this keyword to specify the number of lines in the AIRSAR image.
GENFAC
Use this keyword to specify an array of COMP SCALE FACTORS for each of the files specified by FNAME.

IN_MEMORY
Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

KX
Set this keyword to specify the X size, in pixels, of the averaging box.

KY
Set this keyword to specify the Y size, in pixels, of the averaging box.

LOOKUP
Use this keyword to specify an array of color tables values for the classification image. Each output class can have a unique color triple \([r, g, b]\), LOOKUP is a byte array of size \((3, \text{num_classes}+1)\). Remember that class zero must also have a color triplet (commonly black \([0,0,0]\)).

METHOD
Set this keyword equal to one of the following values to specify the type of processing to perform.

- 0- Perform scattering computations only.
- 1- Classify the data based on the scattering mechanism.

For a Minimum rule decision, the class (or band) with the smallest value becomes the selected class.

OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.
OFFSET
Use this keyword to specify a long array of header offsets for each of the files specified by FNAME.

RULE_OUT_BNAME
Use this keyword to specify a string array that contains the output band names for the rule image.

RULE_OUT_NAME
Use this keyword to specify an output filename for the rule image. If this item is present, the rule image is automatically saved.

RULE_IN_MEMORY
Set this keyword to specify that output rule images should be stored in memory.

RULE_R_FID
Use this keyword to specify a named variable that will contain the file ID for the rule image. This file ID can be used to access the data.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

XSTART
Use this keyword to specify the X offset of the input file. Set to 0 for no offset.

YSTART
Use this keyword to specify the Y offset of the input file. Set to 0 for no offset.

Example

```
pro example_airsar_scatter_doit

; First restore all the base save files.
; envi, /restore_base_save_files
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
```
environ_init, log_file='batch.txt'
;
; Set the keywords
;
fname = ['indvc.dat', $  
        'indvl.dat', $  
        'indvp.dat']

fns = 1024
fnl = 1224
out_type = 0
out_name = 'testimg'
offset = [30720, 30720, 30720]
genfac = [18.1776, 6.51427, 1.63261]
dims = [-1, 0, fns-1, 0, fnl-1]
class_names = [$  
    'Unclassified', $  
    'No vegetation', $  
    'Low vegetation', $  
    'Dihedral Low vegetation', $  
    'Forest', $  
    'Dihedral Forest', $  
    'Medium Vegetation', $  
    'Dihedral Medium Vegetation', $  
    'Urban']
lookup = [$  
    [0, 0, 0], [255, 0, 0], $  
    [0, 255, 0], [0, 0, 255], $  
    [255, 255, 0], [255, 0, 255], $  
    [0, 255, 255], [128, 255, 0], $  
    [128, 0, 255]]
;
; Call the "doit"
;
environ_doit, 'airsar_scatter_doit', $  
    fname=fname, offset=offset, $  
    fns=fns, fnl=fnl, dims=dims, $  
    kx=3, ky=3, class_name=class_names, $  
    lookup=lookup, method=1, xstart=0, $  
    ystart=0, out_name=out_name, $  
    genfac=genfac, r_fid=r_fid
;
; Exit ENVI
;
environ_batch_exit
end
**AIRSAR_SYNTH_DOIT**

Use this program to synthesize an AIRSAR image from a compressed stokes matrix.

### Calling Sequence

ENVI_DOIT, ‘AIRSAR_SYNTH_DOIT’

### Keywords

**COMBO**

Use this keyword to specify a 5 x n array of ellipticity and orientation angles for each image synthesized. The format for the array is:

- (0,i) - transmit ellipticity for i\(^{th}\) image.
- (1,i) - transmit orientation for i\(^{th}\) image.
- (2,i) - receive ellipticity for i\(^{th}\) image.
- (3,i) - receive orientation for i\(^{th}\) image.
- (4,i) - stokes band number 0-C, 1-L, 2-P.

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**DO_SQRT**

Use this keyword to specify that the square root of the data be performed before converting to byte. This keyword is used when the output data type is 1 (BYTE).
FNAME
Use this keyword to specify a three element string array of compressed stokes matrix file names for C, L and/or P wavelengths, respectively. If a file is not used, set the array element to "."

FNS
Use this keyword to specify the number of samples per line in the AIRSAR image.

FNL
Use this keyword to specify the number of lines in the AIRSAR image.

GENFAC
Use this keyword to specify an array of COMP SCALE FACTORS for each of the files specified by FNAME.

IN_MEMORY
Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

MAX_VAL (optional)
Use this keyword to set an optional maximum value for output data.

OFFSET
Use this keyword to specify a long array of header offsets for each of the files specified by FNAME.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

OUT_DT
Use this keyword to specify the output data type, either 1 for byte or 4 for floating point. All other output data types are invalid.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.
SIGMA_ZERO
Set this keyword to specify that the output values be converted to sigma zero, 10*log10(data).

STDMULT
Set this keyword to specify the standard deviation multiplier for byte output data types. Plus and minus the STDMULT determines the output minimum and maximum.

TOTAL_POWER
Use this keyword to specify a three element array of ones and zeros indicating whether the total power should be computed for the C, L and/or P wavelengths, respectively. A value of one causes the synthesis of the total power image.

XFAC
Use this keyword to specify an X subsampling factor used to compute image data statistics prior to the conversion to byte. This keyword is used when the output data type is byte.

YFAC
Use this keyword to specify a Y subsampling factor used to compute image data statistics prior to the conversion to byte. This keyword is used when the output data type is byte.

Example

```plaintext
pro example_airsar_synth_doit
 ;
 ; First restore all the base save files.
 ;
 envi, /restore_base_save_files
 ;
 ; Initialize ENVI and send all errors
 ; and warnings to the file batch.txt
 ;
 envi_batch_init, log_file='batch.txt'
 ;
 ; Set the keywords
 ;
 fname = ['indvc.dat',$
        'indvl.dat',$
        'indvp.dat']
```
fns = 1024
fnl = 1224
out_type = 0
out_name = 'testimg'
offset = [30720, 30720, 30720]
genfac = [18.1776, 6.51427, 1.63261]
dims = [-1, 0, fns-1, 0, fnl-1]
combo = [[0.,0.,0.,0.,0.], $
[0.,0.,0.,0.,1.], $
[0.,0.,0.,0.,2.]]
total_power = lonarr(3) + 1
;
; Call the "doit"
;
envi_doit, 'airsar_synth_doit', $
fname=fname, offset=offset, $
fns=fns, fnl=fnl, dims=dims, $
combo=combo, out_name=out_name, $
total_power=total_power, out_dt=4, $
genfac=genfac, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
ASPECT_DOIT

Use this program to make aspect corrections to Landsat MSS image data.

Calling Sequence

ENVI_DOIT, ‘ASPECT_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.
POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

pro example_aspect_doit
  ; First restore all the base save files.
  ; envi, /restore_base_save_files
  ; Initialize ENVI and send all errors and warnings to the file batch.txt
  ; envi_batch_init, log_file='batch.txt'
  ; Open the input file
  ; envi_open_file, 'bhtmref.img', r_fid=fid
  if (fid eq -1) then begin
    envi_batch_exit
    return
  endif
  ; Set the keywords. We will perform the aspect correction on all samples and bands in the file.
  ; envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
  ; Perform the aspect correction
  ; envi_doit, 'aspect_doit', $
  fid=fid, pos=pos, dims=dims, $
  out_name=out_name, r_fid=r_fid
  ; Exit ENVI
; envi_batch_exit
end
AUTO_WID_MNG

Use this function to automatically perform event handling of ENVI compound widgets, without the need to write an event-handler procedure. The function returns an anonymous structure whose tag names are defined by the user values (uvalue) of the widgets being managed. Each field contains the data item(s) listed under the heading Widget Event for the compound widgets described in this section. AUTO_WID_MNG automatically creates an OK and Cancel button on the widget unless the optional keyword NO_BUTTONS is set. In all cases, if the OK button is selected, the field result.accept (where result is the name of the structure returned by AUTO_WID_MNG) is set to one. Otherwise, if the Cancel button is selected then result.accept is set to zero.

Calling Sequence

Result = AUTO_WID_MNG (Base)

Arguments

Base

The widget ID of the base widget.

Keywords

COLUMN_BASE (optional)

Set this optional keyword to the widget ID of the base that will hold the Accept and Cancel buttons. The default is to use the base widget specified by the Base argument.

NO_BUTTONS (optional)

Set this optional keyword to cause the widget to exit on the first event.
BAD_DATA_DOIT

Use this program to remove bad data lines from image data by averaging adjacent lines.

Calling Sequence

ENVI_DOIT. ‘BAD_DATA_DOIT’

Keywords

BAD_LINES

Use this keyword to specify an array of line numbers to replace. Note that line numbers start with zero. BAD_LINES is a long integer array.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

WIDTH

Use this keyword to specify how many lines above and below the bad line to use in averaging. The total number of lines averaged is 2 * WIDTH. WIDTH is a long integer greater than or equal to one.

Example

```pro
pro example_bad_data_doit

; First restore all the base save files.
envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
envi_batch_init, log_file='batch.txt'

; Open the input file
envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif

; Set the keywords. We will perform the aspect correction on all samples and bands in the file. We will correct lines 100, 120 and 167 for bad data (these lines actually
BAD_DATA_DOIT

BAD_DATA_DOIT

envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
post = lindgen(nb)
out_name = 'testimg'
bads_lines = [100L, 120, 167]

; Perform the bad data correction.
; Use a width of one to correct
; for the bad lines.

envi_doit, 'bad_data_doit', 
    fid=fid, pos=pos, dims=dims, 
    out_name=out_name, width=1, 
    bad_lines=bads_lines, r_fid=r_fid

; Exit ENVI

end
CATCH

The CATCH procedure provides a generalized mechanism for the handling of exceptions and errors. Calling CATCH establishes an error handler for the current procedure that intercepts all errors that can be handled by IDL, excluding non-fatal warnings such as math errors.

When an error occurs, each active procedure, beginning with the offending procedure and proceeding up the call stack to the main program level, is examined for an error handler. If an error handler is found, control resumes at the statement after the call to CATCH. The index of the error is returned in the argument to CATCH. The !ERROR (or !SYSERROR) and !ERR_STRING (or !SYSERR_STRING) system variables are also set. If no error handlers are found, program execution stops, an error message is issued, and control reverts to the interactive mode. A call to ON_IOERROR in the procedure that causes an I/O error supersedes CATCH, and takes the branch to the label defined by ON_IOERROR.

This mechanism is similar, but not identical to, the setjmp/longjmp facilities in C and the catch/throw facilities in C++.

Calling Sequence

CATCH, Variable

Arguments

Variable

A named variable in which the error index is returned. When an error handler is established by a call to CATCH, Variable is set to zero. If an error occurs, Variable is set to the error index and control is transferred to the statement after the call to CATCH.

Keywords

CANCEL

Set this keyword to cancel the error handler for the current procedure. This cancellation does not affect other error handlers that may be established in other active procedures.
CF_DOIT

Use this program to create an output file to disk or memory from bands in the Available Bands list.

Calling Sequence

ENVI_DOIT, ‘CF_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify an array of file IDs, one for each band in the POS array.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.
Chapter 9: ENVI Routines

OUT_DT
Use this keyword to specify a named variable that will contain the IDL data type of the file, using the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify an array of band positions, one for each file ID in the FID array.

REMOVE
Set this keyword to delete input files that are completely contained in the new output file from either disk or memory. WARNING: use this keyword with great caution; files deleted in this manner cannot be restored.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

pro example_cf_doit
  ; First restore all the base save files.
  ; envi, /restore_base_save_files
  ; Initialize ENVI and send all errors
  ; and warnings to the file batch.txt
  ; envi_batch_init, log_file='batch.txt'
  ; Open the input file
  ; envi_open_file, 'bhtmref.img', r_fid=fid
  if (fid eq -1) then begin


envi_batch_exit
return
endif

; Set the keywords. We will perform the
; aspect correction on all samples
; and bands in the file.
;
envi_file_query, fid, ns=ns, nl=nl1, nb=nb
t_fid = lonarr(nb) + fid
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'

; Create the new output file. Do not
; remove the input file after the
; new file has been created.
;
envi_doit, 'cf_doit', $
  fid=t_fid, pos=pos, dims=dims, $
  remove=0, out_name=out_name, $
  r_fid=r_fid

; Exit ENVI
;
envi_batch_exit
end
CLASS_CONFUSION_DOIT

Use this program to compute the confusion matrix, commission, omission, accuracy and kappa coefficient, between a classification image and its ground truth. Ground truth can be specified as either a classification image using the keyword GTFID, or as ROI using ROI_IDS keyword.

Calling Sequence

ENVI_DOIT, ‘CLASS_CONFUSION_DOIT’

Keywords

ACCURACY (optional)

Use this keyword to specify a named variable that will contain the accuracy between the classification image and the ground truth.

CALC_PERCENT

Set this keyword equal to one of the following values to specify either units for the confusion matrix:

- 0 - Output the confusion matrix in pixel counts
- 1 - Output the confusion matrix in percent.
- 2 - Output both types of confusion matrix.

CFID

Use this keyword to specify the file ID for the classification file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

CLASS_PTR

Use this keyword to specify which classes in the classification image match the ground truth ROIs or the ground truth classes specified by GT_PTR. The number of elements of CLASS_PTR must equal the number of ROIs when using ROIs, or the number of elements of CLASS_PTR must equal the number of elements of GT_PTR when using a ground truth classification image.
**COMMISSION** (optional)

Use this keyword to specify a named variable that will contain the commission array between the classification image and the ground truth.

**CPOS**

Use this keyword to specify the band position for the classification image band. For classification images POS should be set to zero. POS is an array of long integers, ranging from zero to the number of bands-1.

**DIMS**

Use this keyword to specify the spatial dimensions of the classification image on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

Note that the dimensions specified must be within the classification image, including the first pixel offsets.

**GT_NAMES**

Use this keyword to specify names for each ground truth class. GT_NAMES is an array of strings with the same number of elements as GT_PTR or ROI_IDS depending on which type of ground truth is used.

**GT_PTR**

Use this keyword to specify which classes in the ground truth image match the classification classes. This keyword is not needed if ROIs are used for ground truth. The number of elements of GT_PTR must equal the number of elements of CLASS_PTR.
GTDIMS
Use this keyword to specify the spatial dimensions of the ground truth image on which to perform the operation. This keyword is not needed if ROIs are used for ground truth. DIMS is a five-element array of long integers with the following definitions:
- DIMS(0): A pointer to the region of interest. Set to -1 for non ROI items.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.
Note that the dimensions specified must be within the classification image, including the first pixel offsets.

GTFID
Use this keyword to specify the file ID for the ground truth file. This keyword is not needed if ROIs are used for ground truth. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

GTPOS
Use this keyword to specify the band position for the ground truth image band. This keyword is not needed if ROIs are used for ground truth. For classification images POS should be set to zero. POS is an array of long integers, ranging from zero to the number of bands-1.

KAPPA_COEFF (optional)
Use this keyword to specify a named variable that will contain the kappa coefficient between the classification image and the ground truth.

MATRIX (optional)
Use this keyword to specify a named variable that will contain the confusion matrix (in pixel counts) between the classification image and the ground truth.

OMISSION (optional)
Use this keyword to specify a named variable that will contain the omission array between the classification image and the ground truth.
ROI_IDS

Use this keyword to specify the ROI ids for the ground truth. This keyword is not needed if a ground truth image is used. Use the keyword GTFID in this case. The number of ROI_IDS must be equal to the number of elements in the CLASS_PTR array.

RPT_COMMISSION (optional)

Set this keyword to one to report the commission to the screen.

RULE_OUT_BNAME (optional)

Use this keyword to specify a string array that contains the output band names for the rule image.

RULE_OUT_NAME (optional)

Use this keyword to specify an output filename for the rule image. If this item is present, the rule image is automatically saved.

RULE_IN_MEMORY (optional)

Set this keyword to specify that output rule images should be stored in memory.

TO_SCREEN

Set this keyword to one to report the confusion matrix to the screen. Otherwise set this value to zero.

Example

```plaintext
forward_function envi_get_roi_ids
pro example_class_confusion_doit
;
; First restore all the base save files.
; envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
; envi_open_file, 'bhtm_class', r_fid=fid
```
if (fid eq -1) then begin
  envi_batch_exit
  return
endif
;
; Restore the ground truth ROIs
;
envi_restore_rois, 'bhtm_nd.roi'
roi_ids = envi_get_roi_ids(fid=fid)
;
; Set the necessary variables
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb, num_classes=num_classes
  pos = [0]
  dims = [-1, 0, ns-1, 0, nl-1]
  out_name = 'testimg'
  class_ptr = lindgen(n_elements(roi_ids)) + 1
  ;
  ; Call the doit
  ;
envi_doit, 'class_confusion_doit',
    cfid=fid, cpos=pos, dims=dims,
    roi_ids=roi_ids, class_ptr=class_ptr,
    /rpt_commission, to_screen=0,
    calc_percent=0, commission=commission,
    omission=omission, matrix=matrix,
    kappa_coeff=kappa_coeff, accuracy=accuracy
  ;
print, commission
print, omission
print, matrix
print, kappa_coeff
print, accuracy
;
; Exit ENVI
;
envi_batch_exit
end
CLASS_CS_DOIT

Use this program to clump or sieve a classification image. The sieve operation uses a blob technique to eliminate all blobs smaller than SIEVE_MIN. The clump process uses the morphological operators dilate and erode.

Calling Sequence

ENVI_DOIT, 'CLASS_CS_DOIT'

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

Note that the dimensions specified must be within the classification image, including the first pixel offsets.

DKERN (Clump only)

Dilate kernel used for the classification Clump. This kernel is passed to the IDL morphology routines.

EIGHT (Sieve only)

If this keyword is set, the sieve function searches the eight-neighbor region around a pixel, rather than the four-neighbor region, for continuous blobs. The four-neighbor region around a pixel consists of two adjacent horizontal and two adjacent vertical neighbors. The eight-neighbor region around a pixel consists of all the immediately adjacent pixels.
EKERN (Clump only)

Erode kernel used for the classification Clump. This kernel is passed to the IDL morphology routines.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**METHOD**

Set this keyword equal to one of the following values to specify either sieve or clump:

- 0 - Clump
- 1 - Sieve

**ORDER**

Use this keyword to specify the order in which clump or sieve is applied to the classification image. If not specified, the classes are processed from first to last.

**OUT_BNAME**

Use this keyword to specify a string array of output band names, if desired.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.
**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**SIEVE_MIN (Sieve only)**

Use this keyword to specify the minimum size of a blob to keep. All blobs smaller than SIEVE_MIN will be removed. Use the EIGHT keyword to specify the pixel neighbors from which to build blobs.

**Example**

```
pro example_class_cs_doit
 ; First restore all the base save files.
 ; envi, /restore_base_save_files
 ; Initialize ENVI and send all errors and warnings to the file batch.txt
 ; envi_batch_init, log_file='batch.txt'
 ; Open the input file
 ; envi_open_file, 'bhtm_sam.img', r_fid=fid
 if (fid eq -1) then begin
   envi_batch_exit
   return
 endif
 ; Set the necessary variables
 ; envi_file_query, fid, ns=ns, nl=nl, $
   num_classes=num_classes
   pos = [0]
   dims = [-1, 0, ns-1, 0, nl-1]
   out_name = 'testimg'
   dkern = bytarr(3,3) + 1b
   ekern = dkern
   order = bindgen(num_classes)
 ; Call the doit
 ; envi_doit, 'class_cs_doit', fid=fid, $
   pos=pos, dims=dims, order=order, $
   dkern=dkern, ekern=ekern, method=0, $
```
out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
CLASS_DOIT

Use this program to perform supervised classification using the Parallelepiped, Minimum Distance, Maximum Likelihood, Spectral Angle Mapper, Mahalanobis, or Binary Encoding methods, or unsupervised classification using ISODATA or K-Means. All classification methods use both the common setup keywords and classification-method specific keywords. The METHOD keyword is used to select the supervised or unsupervised classification technique.

Calling Sequence

ENVI_DOIT, ‘CLASS_DOIT’

Keywords

CLASS_NAMES

Use this keyword to specify names for each output class for the supervised classification methods (the unsupervised methods generate their own CLASS_NAMES based on the color of the class). CLASS_NAMES is an array of strings with num_classes+1 elements. The first element of CLASS_NAMES is the unclassified class and is typically called “Unclassified.” The order of the other classes is determined by the order of the classification data in the keyword MEAN.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.
IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

LOOKUP

Use this keyword to specify a long array of class RGB values for the supervised methods only (the unsupervised methods generate their own LOOKUP based on the number of output classes). The LOOKUP array contains an RGB triplet for the Unclassified class plus one RGB triple for each output class. The Unclassified class, class zero, typically uses the RGB triplet [0,0,0] for black. The dimensions of the array are [3,num_classes+1] and the RGB triplet is ordered [R,G,B]. Class zero, LOOKUP[*,0], is the Unclassified class and the order of the other classes is determined by the order of the classification data in keyword MEAN.

M_FID (optional)

Use this keyword to specify the file ID for the mask file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

M_POS (optional)

Use this keyword to specify the band position of the mask band. M_POS is a single long value greater than or equal to zero.

METHOD

Set this keyword equal to one of the following values to specify the classification method:

- 0 – Parallelepiped (Supervised)
- 1 - Minimum Distance (Supervised)
- 2 - Maximum Likelihood (Supervised)
- 3 - Spectral Angle Mapper (Supervised)
- 4 – Isodata (Unsupervised)
- 5 – Mahalanobis (Supervised)
- 6 - Binary Encoding (Supervised)
- 7 - K-Means (Unsupervised)
OUT_BNAME (optional)

Use this optional keyword to specify an output band name. OUT_BNAME is a single string value representing the classification image band name.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID (optional)

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Keywords for Supervised Classification

The following keywords can be used with any of the supervised classification methods (parallelepiped, minimum distance, maximum likelihood, Mahalanobis distance spectral angle mapper, and binary encoding):

MEAN

Use this keyword to specify the mean spectral values for each class when performing supervised classification. MEAN is a floating-point or double-precision array of [nb,num_classes] values. The spectral mean of each class (for supervised methods) is commonly computed from the spectral mean of the ROI representing the training region of the class. The actual number of output classes, NUM_CLASSES, is computed from the number of spectral means plus one for the Unclassified class.

Note

For the unsupervised methods of ISODATA and K-Means, the initial starting classes are calculated automatically from the mean on the input data and do not require the MEAN keyword.
RULE_FID (optional)
Use this keyword to specify a named variable that will contain the file ID for the processed rule image. This file ID can be used to access the processed data.

RULE_OUT_BNAME (optional)
Use this keyword to specify a string array that contains the output band names for the rule image.

RULE_OUT_NAME (optional)
Use this keyword to specify an output filename for the rule image. If this item is present, the rule image is automatically saved.

RULE_IN_MEMORY (optional)
Set this keyword to specify that output rule images should be stored in memory.

Keywords for Parallelepiped Classification

STDV
Use this keyword to specify a floating-point or double-precision array of the form \((num\_bands, num\_classes)\) containing the standard deviation for each of the spectral classes.

STD_MULT (optional)
Use this optional keyword to specify a floating-point or double-precision multiplication factor or array of factors (one for each class) specifying the width around the standard deviation within which spectrum may fall and still be classified into that class. If an array is specified, each class is tested with its corresponding width. The default is 1.0.

Keywords for Minimum Distance Classification

STDV
Use this keyword to specify a floating-point or double-precision array of the form \((num\_bands, num\_classes)\) containing the standard deviation for each of the spectral classes. This value must be specified if STD_MULT is specified.
STD_MULT (optional)

Use this optional keyword to specify a floating-point or double-precision multiplication factor or array of factors (one for each class) specifying the width around the standard deviation within which spectrum may fall and still be classified into that class. If an array is specified, each class is tested with its corresponding width. The default is 1.0. If SDT_MULT is specified, the keyword STDV must be set.

THRESH (optional)

Use this optional keyword to specify a floating-point or double-precision maximum distance error or array of errors (one for each class) by which a spectral value can differ from the mean value. If an array is specified, each class is tested against its corresponding error. When a single value is specified, THRESH is applied class by class, not as the total error.

Keywords for Maximum Likelihood Classification

COV

Use this keyword to specify a floating-point or double-precision array of the form (num_bands, num_bands, num_classes) containing the covariance of the classification spectrum used.

STDV

Use this keyword to specify a floating-point or double-precision array of the form (num_bands, num_classes) containing the standard deviation for each of the spectral classes.

THRESH (optional)

Use this optional keyword to specify a floating-point or double-precision minimum probability or array of probabilities (one for each class) that a class must have in order to be classified. If an array is specified, each class is tested against its corresponding probability.

Keywords for Spectral Angle Mapper Classification

THRESH (optional)

Use this optional keyword to specify a floating-point or double-precision maximum angle or array of maximum angles (one for each class) specifying the maximum angle to classify (in radians). If an array is specified, each class is tested against its
corresponding maximum spectral angle. THRESH should have a value between zero and $\pi/2$.

**Keywords for Mahalanobis**

**COV**

Use this keyword to specify a floating-point or double-precision array of the form $(\text{num\_bands}, \text{num\_bands}, \text{num\_classes})$ containing the covariance of the classification spectrum used.

**THRESH (optional)**

Use this optional keyword to specify a floating-point or double-precision maximum distance error or array of errors (one for each class) by which a spectral value can differ from the mean value. If an array is specified, each class is tested against its corresponding error. When a single value is specified, THRESH is applied class by class, not as the total error.

**Keywords for Binary Encoding**

**THRESH (optional)**

Use this optional keyword to specify floating-point or double-precision minimum match percent or array of minimum match percents (one for each class). The value of THRESH is between 0.0 and 1.0 ranging from 0% to 100%. If an array is specified, each class is tested against its corresponding minimum match percent. For example, a value of 1.0 means that all bands must match and a value of .4 means that at least 40% of the bands must match the binary encoding.

**Keywords for Unsupervised Classification**

The following keywords can be used with any of the unsupervised classification methods (K-Means and ISODATA):

**STD_MULT (optional)**

Use this keyword to specify a floating-point multiplication factor specifying the width around the standard deviation within which a spectrum may fall and still be classified into that class. The default is 1.0.
THRESH (optional)

Use this keyword to specify the maximum distance error by which a spectral value can differ from a mean value. The THRESH value is applied band by band, not as the total error.

ITERATIONS

Use this keyword to specify the maximum iteration count.

Keywords for K-Means

CHANGE_THRESH

Set this keyword to a floating-point number between zero and one to specify the percentage of pixels which can change classes during each iteration. If the percentage of pixels that will change in a given iteration is greater than the CHANGE_THRESH value, another iteration is performed, provided that the maximum number of iterations is not exceeded. If the percentage is less then the threshold, the classification is complete. A value of 1.0 means 100%.

NUM_CLASSES

Use this keyword to specify the desired number of output classes.

Keywords for ISODATA

ISO_MERGE_DIST

Set this keyword to a floating-point number greater than zero to specify the class merge distance (in DN). If the distance between class means is less than ISO_MERGE_DIST, the classes will be merged. The maximum number of pairs merged in any loop is determined by ISO_MERGE_PAIRS

ISO_MERGE_PAIRS

Set this keyword to a long value to specify the maximum number of classes that can be merged in a single iteration.

ISO_MIN_PIXELS

Set this keyword to a long value to specify the minimum number of pixels needed to form a class. If there are few pixels in the class, that class will be deleted.
ISO_SPLIT_STD

Set this keyword to a floating-point number greater than zero to specify the minimum class standard deviation value (in DN). If a class standard deviation is greater than ISO_SPLIT_STD, the class is split into two classes.

ISO_SPLIT_SMULT (optional)

Set this keyword to a floating-point number greater than zero to specify the standard deviation multiplier used to calculate the mean of split classes. The new means are calculated as follows

\[
\text{class}_1\text{.mean} = \text{class\_mean} + \text{ISO\_SPLIT\_STD} \times \text{current\_stdv} \\
\text{class}_2\text{.mean} = \text{class\_mean} - \text{ISO\_SPLIT\_STD} \times \text{current\_stdv}
\]

The default value is 1.

MIN_CLASSES

Use this keyword to specify the desired minimum number of output classes.

NUM_CLASSES

Use this keyword to specify the desired maximum number of output classes.

Example

This example performs a supervised Maximum Likelihood classification using ROIs previously saved to an ROI file. The Maximum Likelihood method requires setting the keywords MEAN, STDV, and COV. The routine ENVI_STATS_DOIT is used to calculate the ROI mean, standard deviation, and covariance. The class names and colors come from the ROI names and colors respectively.

This example uses the following files found on the ENVI Tutorial and Data CD No. 1:

- `envidata/bh_tmsub/bhtm_mnf.img`
- `envidata/bh_tmsub/bhtm_mnf.hdr`
- `envidata/bh_tmsub/bhtm_nd.roi`

```plaintext
forward_function envi_get_roi_ids, envi_get_roi_dims_ptr
pro example_class_doit
    ;
    ; First restore all the base save files.
    ;
    envi, /restore_base_save_files
```
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
;
; Open the input file
; envi_open_file, 'bhtm_mnf.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Get the samples, lines and # bands
; for the input file.
;
; envi_file_query, fid, ns=ns, nl=nl, nb=nb
;
; Set the dims and pos to classify all
; data (spatially and spectrally) in the file.
;
; dims = [-11, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
;
; Restore the pre saved regions of interest.
; Set the class names equal to the roi
; names and use the roi colors as the
; class colors.
;
; envi_restore_rois, 'bhtm_nd.roi'
roi_ids = envi_get_roi_ids(fid=fid,$
    roi_colors=roi_colors, roi_names=class_names)
class_names = ['Unclassified', class_names]
num_classes = n_elements(roi_ids)
lookup = bytarr(3, num_classes + 1)
; Set the unclassified class to black and use roi colors
lookup = bytarr(3,num_classes+1)
lookup[0,1] = roi_colors
;
; Calculate the statistics from each region of interest.
; Each ROI specifies the training area for each class.
; The calculated MEAN, STDV and COV will be passed
; to the classification routine.
;
mean = fltarr(n_elements(pos), num_classes)
stdv = fltarr(n_elements(pos), num_classes)
cov = fltarr(n_elements(pos),n_elements(pos),num_classes)
for j=0, num_classes-1 do begin
    ; get the statistics for each selected class
    roi_dims=[envi_get_roi_dims_ptr(roi_ids[j]),0,0,0,0]
    envi_doit, 'envi_stats_doit', fid=fid, pos=pos, $
    \quad \text{dims}=roi\_dims, \text{comp\_flag}=4, \text{mean}=c\_mean, \$
    \quad \text{stdv}=c\_stdv, \text{cov}=c\_cov
    mean[0,j] = c\_mean
    stdv[0,j] = c\_stdv
    cov[0,0,j] = c\_cov
endfor

; Calculate the Maximum Likelihood supervised
; classification.

; envi_doit, 'class\_doit', fid=fid, pos=pos, dims=dims, $
; \quad \text{out\_bname}='Max Like', \text{out\_name}=out\_name, \text{method}=2, \$
; \quad \text{mean}=mean, \text{stdv}=stdv, \text{std\_mult}=st\_mult, r\_fid=r\_fid, $
; \quad \text{lookup}=lookup, \text{cov}=cov, \text{class\_names}=class\_names, \$
; \quad \text{num\_classes}=num\_classes, \text{in\_memory}=0

; Exit ENVI

envi_batch_exit
end
CLASS_MAJORITY_DOIT

Use this program to perform majority or minority analysis on a classification image. The keyword CLASS_PTR determines which classes to modify during processing.

Calling Sequence

ENVI_DOIT, ‘CLASS_MAJORITY_DOIT’

Keywords

CLASS_PTR

Use this keyword to specify a long-integer array of class numbers on which to perform analysis.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
KERNEL_SIZE

Use this keyword to specify a kernel size for the majority and minority analysis. KERNEL_SIZE is a two element array specifying the X and Y kernel size, respectively.

METHOD

Set this keyword equal to one of the following values to specify the analysis method:

- 0 - Majority
- 1 - Minority

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID (optional)

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_class_majority_doit
;
; First restore all the base save files.
;
envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
;
envi_batch_init, log_file='batch.txt'
;
; Open the input file
```
; envi_open_file, 'bhtm_sam.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Set the necessary variables
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb, $
    num_classes=num_classes
pos = [0]
dims = [-1l, 0, ns-1, 0, nl-1]
out_name = 'testimg'
method = 0
kernel_size = [5,5]
class_ptr = lonarr(num_classes) + 1
;
; Call the doit
;
envi_doit, 'class_majority_doit', fid=fid, $
    pos=pos, dims=dims, method=method, $
    kernel_size=kernel_size, out_name=out_name, $
    class_ptr=class_ptr
;
; Exit ENVI
;
envi_batch_exit
end

See Also

CLASS_CS_DOIT, CLASS_DOIT
CLASS_RULE_DOIT

Use this program to classify rule images.

Calling Sequence

ENVI_DOIT, ‘CLASS_RULE_DOIT’

Keywords

CLASS_NAMES

Use this keyword to specify names for each output class. CLASS_NAMES is an array of strings with num_classes+1 elements. Remember to set the zero class to “Unclassified”.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

Note that the dimensions specified must be within the classification image, including the first pixel offsets.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
LOOKUP

Use this keyword to specify an array of color table values for the classification image. Each output class can have a unique color triple \([r, g, b]\), LOOKUP is a byte array of size \((3, \text{num\_classes}+1)\). Remember that class zero must also have a color triplet (commonly black \([0,0,0]\)).

METHOD

Set this keyword equal to one of the following values, to specify either a minimum or maximum rule decision:

- 0 - Minimum
- 1 - Maximum

For a Minimum rule decision, the class (or band) with the smallest value becomes the selected class.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN\_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

THRESH (optional)

Use this keyword to specify an optional minimum (METHOD=0) or maximum (METHOD = 1) threshold for classifying the rule image. If no threshold is specified, the entire image will be classified.

Example

```
pro example_class_rule_doit

; First restore all the base save files.

envi, /restore_base_save_files
```
; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; Open the input file
; envi_open_file, 'bhtm_rul.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
; Set the necessary variables
; envi_file_query, fid, ns=ns, nl=nl1, $ nb=nb, bnames=bnames
pos = lindgen(nb)
dims = [-1, 0, ns-1, 0, nl-1]
out_name = 'testimg'
class_names = ['Unclassified', bnames]
; Build the lookup table
; lookup = bytarr(3,nb+1)
for i=1,nb do begin
    envi_get_rgb_triplets, i+1, r, g, b
    lookup[0,i] = [r,g,b]
endfor
; Call the doit
; envi_doit, 'class_rule_doit', $ fid=fid, pos=pos, dims=dims, $ lookup=lookup, method=0, $ class_names=class_names, $ out_name=out_name, r_fid=r_fid
; Exit ENVI
; envi_batch_exit
end
CLASS_STATS_DOIT

Use this program to calculate statistics for images based on classification mask.

Calling Sequence

ENVI_DOIT, ‘CLASS_STATS_DOIT’

Keywords

BIN SIZE
Use this keyword to set the bin size used for histograms. BIN SIZE is used with floating point data.

CLASS_FID
Use this keyword to specify the file ID of the classification image.

CLASS_PTR
Use this keyword to specify a long-integer array of class numbers on which to perform statistics.

COMP_FLAG
Set this keyword equal to a bit value indicating the computations to perform.

• bit 0 - not used.
• bit 1 - enables the calculation of histograms.
• bit 2 - enables the calculation of covariance.
• bits 3 to 15 - not used.

COV
Use this keyword to specify a named variable that will contain the returned covariance matrix. You must set bit 2 in COMP_FLAG (i.e. COMP_FLAG=4) to generate the covariance matrix.

DIMS
Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:
Chapter 9: ENVI Routines

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

Note that the dimensions specified must be within the classification image, including the first pixel offsets.

**DMAX**

Use this keyword to specify a named variable that will contain the array of data maximums, one for each band position.

**DMIN**

Use this keyword to specify a named variable that will contain the array of data minimums, one for each band position.

**EVAL**

Use this keyword to specify a named variable that will contain the eigenvalues. You must set bit 2 in COMP_FLAG (i.e. COMP_FLAG=4) to generate the covariance matrix.

**EVEC**

Use this keyword to specify a named variable that will contain the eigenvector. You must set bit 2 in COMP_FLAG (i.e. COMP_FLAG=4) to generate the covariance matrix.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**HIST**

Use this keyword to specify a named variable that will contain the histogram array. You must set bit 1 in COMP_FLAG (i.e. COMP_FLAG=2) to generate the histogram output.
MEAN

Use this keyword to specify a named variable that will contain the array of data means, one for each band position.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

REP_NAME

Use this keyword to specify a string array containing the file name for the output report file.

REPORT_FLAG

Set this keyword equal to a bit value indicating the type of output reports desired. Logically AND the bit values together to get the desired reports.

- bit 0 - basic statistics. value=1.
- bit 1 - generate histogram report (the default), value=2.
- bit 2 - generate covariance report, value=4.
- bits 3 to 15 - not used.

If report flag is zero, no output report is generated.

STA_NAME

Use this keyword to specify a string array containing the file name for the output statistics file.

STDV

Use this keyword to specify a named variable that will contain the array of data standard deviations, one for each band position.

TO_SCREEN

Set this keyword to print the report on the screen.

Example

```pro
pro example_class_stats_doit
```

; First restore all the base save files.
;
envi, /restore_base_save_files
;
; Initialize ENVI and send all errors and warnings to the file batch.txt
;
envi_batch_init, log_file='batch.txt'
;
; Open the data and class files
;
envi_open_file, 'bhtmref.img', r_fid=fid
evi_open_file, 'bhtm_sam.img', r_fid=cfid
if (fid eq -1 or cfid eq -1) then begin
envi_batch_exit
  return
endif
;
; Set the necessary variables
;
envi_file_query, cfid, num_classes=num_classes
class_ptr = indgen(num_classes)
envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1l, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'class_stats.txt'
;
; Call the doit
;
envi_doit, 'class_stats_doit', fid=fid, pos=pos, $
dims=dims, comp_flag=0, report_flag=1, $
rep_name=out_name, class_fid=cfid, $
class_ptr=class_ptr
;
; Exit ENVI
;
envi_batch_exit
 end

See Also

ENVI_STATS_DOIT, ENVI_GET_STATISTICS, ENVI_WRITE_STATISTICS
COM_CLASS_DOIT

Use this program to combine classes in ENVI classified images.

Calling Sequence

ENVI_DOIT, ‘COM_CLASS_DOIT’

Keywords

COMB_LUT

Use this keyword to specify a long-integer array indicating the output class for each input class. Each element in the array indicates the output class for the current input class. COMB_LUT has num_classes elements.

For example, if you wanted to map class 0 to 0, class 1 to 0, class 2 to 3, and class 3 to 2, you would use the following keyword specification:

   comb_lut = [0, 0, 3, 2]

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

REMOVE_EMPTY

Set this keyword to remove empty classes after combining. If empty classes are not removed, one or more classes with zero elements may exist in the output image. The default is to leave the empty classes.

Example

```plaintext
pro example_com_class_doit

; First restore all the base save files.
; envi, /restore_base_save_files
; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; Open the input file
; envi_open_file, 'bhtm_sam.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
; Set the necessary variables
; envi_file_query, fid, ns=ns, nl=n1, $
    num_classes=num_classes
pos  = [0]
dims = [-1, 0, ns-1, 0, nl-1]
out_name = 'testimg'
comb_lut = lindgen(num_classes)
comb_lut[1] = 0
comb_lut[4] = 2
comb_lut[5] = 2
```
; Call the doit
;
envi_doit, 'com_class_doit', $
   fid=fid, pos=pos, dims=dims, $
   comb_lut=comb_lut, /remove_empty, $
   out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
CONTINUUM_REMOVE_DOIT

Use this program to perform continuum removal on the data.

Calling Sequence

ENVI_DOIT, ‘CONTINUUM_REMOVE_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:
- DIMS(0): Set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

M_FID

Use this keyword to specify the file ID for the mask file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

M_POS

Use this keyword to specify the band position of the mask band. M_POS is a single long value greater than or equal to zero.
OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify a one-dimension array of band positions indicating the band numbers to perform the operations on. POS is a long array ranging from 0 to the number of bands-1.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_continuum_remove_doit
;
; First restore all the base save files.
; envi, /restore_base_save_files
;
; Initialize ENVI and send all errors and warnings to the file batch.txt
;
; envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
; envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Set the keywords. We will perform the continuum removal on all samples and bands in the file.
;
; envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
```
out_name = 'testimg'
;
; Perform the continuum removal
;
envi_doit, 'continuum_remove_doit', $
   fid=fid, pos=pos, dims=dims, $
   out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
CONV_DOIT

Use this program to perform convolution filtering, including high pass, low pass, laplacian, gaussian, median, directional, sobel, roberts, and user-defined. For methods 1 and 2 the output data type is automatically set to INTEGER (IDL type = 2) and for the remaining methods the output data type is converted as follows: Input BYTE(1) converts to INTEGER(2), input UNSIGNED INTEGER (12) converts to INTEGER(2), input UNSIGNED LONG (13) converts to LONG (3).

Calling Sequence

ENVI_DOIT, ‘CONV_DOIT’

Keywords

ADD_BACK

Set this keyword equal to a floating-point or double-precision number between 0.0 and 1.0, specifying the filter add-back percentage. A value of 1.0 is equal to 100%.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
KERNEL

Use this keyword to specify a convolution kernel for methods 4 through 8.

KX

Use this keyword to specify the X kernel size. For methods 0 through 3, KX must be equal to KY. For methods 0 and 1, KX must be set equal to three.

KY

Use this keyword to specify the Y kernel size. For methods 0 through 3, KY must be equal to KX. For methods 0 and 1, KY must be set equal to three.

METHOD

Use this keyword to specify the type of filter to apply. Choose one of the following:

- 0 - Sobel
- 1 - Roberts
- 2 - Median
- 3 - Low pass
- 4 - High pass
- 5 - Laplacian
- 6 - Directional
- 7 - Gaussian
- 8 - User defined

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.
**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**Example**

```plaintext
pro example_conv_doit

; First restore all the base save files.
envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
envi_batch_init, log_file='batch.txt'

; Open the data and class files
envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
  envi_batch_exit
  return
endif

; Set the necessary variables
envi_file_query, fid, ns=ns, nl=nl, nb=nb, $
  bname=bname
dims = [-1l, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
method = 2

; Call the doit
envi_doit,'conv_doit', fid=fid, pos=pos, $
  dims=dims, out_name=out_name, $
  out_bname=bname(pos), method=method, $
  kx=5, ky=5, add_back=.2, in_memory=0, $
  r_fid=r_fid

; Exit ENVI
envi_batch_exit
end
```

**CONV_DOIT**

*ENVI Programmer's Guide*
CONVERT_DOIT

Use this program to convert interleave type between BSQ, BIL, and BIP.

Calling Sequence

ENVI_DOIT, ‘CONVERT_DOIT’

Keywords

DIMS
Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID
Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

O_INTERLEAVE
Set this keyword equal to zero to specify BSQ interleave output, one to specify BIL output, or two to specify BIP output.

OUT_NAME
Use this keyword to specify an output file name for the resulting data.

POS
Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.
**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**Example**

```plaintext
pro example_convert_doit
 ; ; First restore all the base save files.
 ; envi, /restore_base_save_files
 ; ; Initialize ENVI and send all errors
 ; and warnings to the file batch.txt
 ; envi_batch_init, log_file='batch.txt'
 ; ; Open the input file
 ; envi_open_file, 'bhtmref.img', r_fid=fid
 if (fid eq -1) then begin
   envi_batch_exit
   return
 endif
 ; ; Set the keywords. We will perform the
 ; continuum removal on all samples
 ; and bands in the file.
 ; envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
 ; ; Perform the file conversion. Convert
 ; the input BSQ image to BIL storage
 ; order.
 ; envi_doit, 'convert_doit', $
   fid=fid, pos=pos, dims=dims, $
o_interleave=1, out_name=out_name, $
r_fid=r_fid
 ; ; Exit ENVI
 ; envi_batch_exit
end
```
CONVERT_INPLACE_DOIT

Use this program to convert, in place, an ENVI file from one storage interleave to another. The program allows conversion between BSQ, BIL, and BIP storage interleaves, without creating temporary files.

Calling Sequence

ENVI_DOIT, ‘CONVERT_INPLACE_DOIT’

Arguments

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

O_INTERLEAVE

Use this keyword to specify the output interleave. Set to zero to convert to Band Sequential (BSQ) format, one to convert to Band Interleave by Line (BIL) format, or two to convert to Band Interleave by Pixel (BIP) format.

POS

Use this keyword to specify a one element array of the band positions to perform the operations on. POS is a single element long array, ranging from 0 to the number of bands-1.
R_FID (optional)

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_convert_inplace_doit
 ; ; First restore all the base save files.
 ; envi, /restore_base_save_files
 ; ; Initialize ENVI and send all errors
 ; and warnings to the file batch.txt
 ; envi_batch_init, log_file='batch.txt'
 ; ; Open the data and class files
 ; envi_open_file, 'bhtmref.img', r_fid=fid
 if (fid eq -1) then begin
   envi_batch_exit
   return
 endif
 ; ; Set the necessary variables
 ; envi_file_query, fid, ns=ns, nl=nl, nb=nb, $
   interleave=interleave
 o_interleave = ([1,2,0])[interleave]
 dims = [-1, 0, ns-1, 0, nl-1]
 pos = lindgen(nb)
 ; ; Call the doit
 envi_doit,'convert_inplace_doit', fid=fid, $
   pos=pos, dims=dims, $
 o_interleave=o_interleave, r_fid=r_fid
 ; ; Exit ENVI
 ; envi_batch_exit
end
```

See Also

CONVERT_DOIT
CROSS_TRACK_CORRECTION_DOIT

Use Cross-Track Illumination Correction to remove variation in the cross-track illumination of an image. Cross track illumination variations may be due to vignetting effects, instrument scanning, or other non-uniform illumination effects. Along-track mean values are calculated and are used to show the mean variation in the cross-track direction. A polynomial function, with a defined order, is fit to the means and used to remove the variation. In addition, CROSS_TRACK_CORRECTION_DOIT is used to perform antenna pattern correction on radar images. Radar images typically have a variation in gain across the range direction (cross-track), due to the instrument’s antenna gain pattern. The Cross-Track provides enough control to perform either the illumination correction or the antenna pattern correction.

Calling Sequence

ENVI_DOIT, ‘CROSS_TRACK_CORRECTION_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY (optional)

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk and OUT_NAME must be specified. If IN_MEMORY is set, OUT_NAME is not used.
M_FID (optional)

Use this optional keyword to specify the file ID for the mask file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1. The mask is used for computing the correction and optionally applied to the output image. See the keyword MASK_OUTPUT.

M_POS (optional)

Use this optional keyword to specify the band position of the mask band. M_POS is a single long value greater than or equal to zero. The mask is used for computing the correction and optionally applied to the output image (see the keyword MASK_OUTPUT).

MASK_OUTPUT (optional)

Set this optional keyword to apply the mask to the output image. By default, the optional mask information (keyword M_FID and M_POS) is applied only when calculating the polynomial correction. If M_FID and M_POS are not set, this keyword has no effect.

METHOD

Set this keyword to one of the following values to specify the correction method.

- 0 – Additive
- 1 – Multiplicative

ORDER (optional)

Use this optional keyword to specify the order of the polynomial for the cross-track illumination or antenna-pattern correction. A polynomial function with the defined order is fit to the means of the data specified by RANGE_DIR and used to remove the variations. The default is “1”, a first degree polynomial.

OUT_BNAME (optional)

Use this optional keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. OUT_NAME is a string variable specifying the filename and path of the output file.
POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID (optional)

Use this optional keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data. An invalid file ID is specified as -1.

RANGE_DIR

Set this keyword to one of the following values, to specify the cross-track or range direction.

- 0 – Samples. The cross-track or range direction is across the samples of a single line
- 1 – Lines. The cross-track or range direction is across the lines of a single sample.

Example

This example is used to compute the cross-track correction for the file bhtmref.img. The correction is applied to all pixels and all bands, without any masking. The cross-track or range direction is set to samples using the RANGE_DIR keyword. The ORDER keyword is set to a second order polynomial and the METHOD keyword is used to perform a multiplicative correction. The output is stored to disk.

This example uses the following files found in the data directory of the ENVI installation:

- bhtmref.img
- bhtmref.hdr

Note

This correction is not scientifically valid—it is applied to this image only to demonstrate the use of the processing routine.

```
pro example_cross_track_correction_doit
    ;
    ; First restore all the base save files.
    ;
    envi, /restore_base_save_files
```
; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
; envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Set the keywords. We will perform the cross track correction on all samples and bands in the file. The correction will be in the samples direction, multiplicative, and with a 2nd order polynomial.
;
; envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1L, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
range_dir = 0
method = 1
order = 2
;
; Perform the cross track correction
;
; envi_doit, 'cross_track_correction_doit', $ fid=fid, pos=pos, dims=dims, $ range_dir=range_dir, out_name=out_name, $ method=method, order=order, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end

See Also

EMITTANCE_CALC_DOIT, ENVI_AVHRR_CALIBRATE_DOIT, ENVI_CAL_DOIT, TIMS_CAL_DOIT, TMCAL_DOIT
DARK_SUB_DOIT

Use this program to do dark region subtraction for atmospheric scattering correction.

Calling Sequence

ENVI_DOIT, ‘DARK_SUB_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.
POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

VALUES

Use this keyword to specify an array of values to subtract from each band of the data. The size of the array is equal to the number of elements in POS.

Example

```plaintext
pro example_dark_sub_doit
 ; First restore all the base save files.
 ; envi, /restore_base_save_files
 ; Initialize ENVI and send all errors
 ; and warnings to the file batch.txt
 ; envi_batch_init, log_file='batch.txt'
 ; Open the input file
 ; envi_open_file, 'can_tmr.img', r_fid=fid
 if (fid eq -1) then begin
   envi_batch_exit
   return
 endif
 ; Set the keywords. We will perform the
 ; continuum removal on all samples
 ; and bands in the file.
 ; envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
values = [10,5,9,7,2,13]
 ; Perform the dark current subtraction.
```
Chapter 9: ENVI Routines

Chapter 9: ENVI Routines

ENVI Programmer's Guide

DARK_SUB_DOIT

ENVI Programmer's Guide

envi_doit, 'dark_sub_doit', $
  fid=fid, pos=pos, dims=dims, $
  values=values, out_name=out_name, $
  r_fid=r_fid

; Exit ENVI

; envi_batch_exit
end
DECOR_DOIT

Use this program to remove high correlation between three bands to make a saturated color image.

Calling Sequence

ENVI_DOIT, ‘DECOR_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify an array of three file IDs for the RGB bands in the decorrelation stretch.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.
POS

Use this keyword to specify an array of three band positions corresponding to the file IDs in the FID array.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_decor_doit
  ;
  ; First restore all the base save files.
  ;
  envi, /restore_base_save_files
  ;
  ; Initialize ENVI and send all errors
  ; and warnings to the file batch.txt
  ;
  envi_batch_init, log_file='batch.txt'
  ;
  ; Open the input file
  ;
  envi_open_file, 'bhtmref.img', r_fid=fid
  if (fid eq -1) then begin
    envi_batch_exit
    return
  endif
  ;
  ; Set the keywords. We will perform the
  ; decorrelation stretch on the first
  ; three bands in the file and all
  ; spatial pixels.
  ;
  envi_file_query, fid, ns=ns, nl=nl
  t_fid = [fid,fid,fid]
  dims = [-1, 0, ns-1, 0, nl-1]
  pos = [0,1,2]
  out_name = 'testimg'
  ;
  ; Perform the decorrelation stretch
  ;
  envi_doit, 'decor_doit',
    fid=t_fid, pos=pos, dims=dims, $
    out_name=out_name, r_fid=r_fid
  ;
```
; Exit ENVI
;
envi_batch_exit
end
DEM_BAD_DATA_DOIT

Use this program to correct bad data points and/or areas in digital elevation data (DEM). Bad data points are defined by a mask band, bad data value, minimum threshold, and/or maximum threshold. The results from these optional keywords are OR’ed together to form the overall bad data mask. At least one of the methods for specifying bad data must be defined.

Calling Sequence

ENVI_DOIT, ‘DEM_BAD_DATA_DOIT’

Keywords

BAD_VALUE (optional)
Use this keyword to specify a bad DEM value. All DEM values equal to BAD_VALUE are considered bad values. BAD_VALUE is a single number.

DIMS
Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID
Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY
Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
M_FID (optional)

Use this keyword to specify the file ID for the bad data mask file. Mask pixels not equal to zero are considered bad data. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer, with a value greater than zero. An invalid file ID is specified as -1.

M_POS (optional)

Use this keyword to specify the band position of the bad data-mask band. Mask pixels not equal to zero are considered bad data. M_POS is a single long value greater than or equal to zero.

MAX_THRESH (optional)

Use this keyword to specify the maximum threshold for bad DEM values. Values less than or equal to this threshold are considered bad data. MAX_THRESH is a single number. When used in conjunction with MIN_THRESH values between both thresholds are considered bad.

MIN_THRESH (optional)

Use this keyword to specify the minimum threshold for bad DEM values. Values greater than or equal to this threshold are considered bad data. MIN_THRESH is a single number. When used in conjunction with MAX_THRESH values between both thresholds are considered bad.

OUT_BNAME (optional)

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.
See Also

TOPO_DOIT
DESKEW_DOIT

Use this program to deskew Landsat MSS data for earth rotation.

Calling Sequence

ENVI_DOIT, ‘DESKEW_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

LATD

Set this keyword to a long integer specifying the latitude degrees for Landsat Scene center.

LATM

Set this keyword to a long integer specifying the latitude minutes for Landsat Scene center.
LATS
Set this keyword to a long integer specifying the latitude seconds for Landsat Scene center.

OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword
IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify an array of band positions, indicating the band numbers
on which to perform the operation. POS is an array of long integers, ranging from
zero to the number of bands-1.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the
processed data. This file ID can be used to access the processed data.

Example

```
pro example_deskew_doit
; ; First restore all the base save files.
; envi, /restore_base_save_files
; ; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; ; Open the input file
; envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
  envi_batch_exit
  return
endif
; ; Set the keywords. We will perform the
; image deskew on all samples and
```
; bands in the file.
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
;
; Deskew the image
;
envi_doit, 'deskew_doit', $
  fid=fid, pos=pos, dims=dims, $
  latd=44, latm=13, lats=0, $
  out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
DESTRIPE_DOIT

Use this program to destripe image data for detector imbalance.

Calling Sequence

ENVI_DOIT, 'DESTRIPE_DOIT'

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

NUM_DET

Use this keyword to specify the number of detectors in the instrument. Set equal to 16 for TM, or equal to 6 for MSS.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```pro
pro example_destripe_doit
    ; First restore all the base save files.
    ; envi, /restore_base_save_files
    ; Initialize ENVI and send all errors
    ; and warnings to the file batch.txt
    ; envi_batch_init, log_file='batch.txt'
    ; Open the input file
    ; envi_open_file, 'bhtmref.img', r_fid=fid
    if (fid eq -1) then begin
        envi_batch_exit
        return
    endif
    ; Set the keywords. We will perform the
    ; destriping on all samples and bands
    ; in the file.
    ; envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
    ; Perform the image destriping
```
envis_doit, 'destripe_doit', $
   fid=fid, pos=pos, dims=dims, $
   num_det=16, out_name=out_name, $
   r_fid=r_fid$

; Exit ENVI
;
envi_batch_exit
end
DISP_GET_LOCATION

Use this routine to return the x and y locations of the current pixel for a given ENVI display. Pixel locations are returned in modified file coordinate system: pixel coord = file coord + (xy)start - 1, where (xy)start is the xstart or ystart value in the ENVI header for the displayed image.

Calling Sequence

DISP_GET_LOCATION

Arguments

DN

The display number corresponding to an open display.

XFLOC

Upon return, this variable contains the x location of the current pixel, in modified file coordinates.

YFLOC

Upon return, this variable contains the y location of the current pixel, in modified file coordinates.

Keywords

FLOATING (optional)

Set this keyword if you want the XFLOC and YFLOC coordinates to be returned as floating point values. If this keyword is not set, then XFLOC and YFLOC will be long integer values.

NO_OFFSET (optional)

Set this keyword to return the current pixel location in ordinary zero-based file coordinates (without (xy) start values being added).

Example

; ; This example prints out the location of ; the current pixel for each display
pro example_disp_get_location

; Get all the display numbers and
; check to make sure at least one
; display is present.
; display = envi_get_display_numbers()
if (display[0] eq -1) then return
; ; Print out the location of the
; current pixel for each display
; for i=0, n_elements(display)-1 do begin
 disp_get_location, display[i], xloc, yloc
 print, display[i], xloc, yloc
 endfor
end

See Also

DISP_GOTO, ENVI_DISP_QUERY
DISP_GOTO

Use this new routine to move the current pixel of a given display. The Zoom window will also be moved, so that it is centered on the new current pixel location. If the new current pixel location is not within the current Image window, the Image and Scroll windows will also be moved, so that they include the new current pixel.

Calling Sequence

DISP_GOTO

Arguments

DN

The display number corresponding to an open display.

XFLOC

Upon return, this variable contains the x location of the new current pixel, in zero-based file coordinates.

YFLOC

Upon return, this variable contains the y location of the new current pixel, in zero-based file coordinates.

Keywords

NO_WARN (optional)

Set this keyword, if you do not want an error message printed when the XFLOC, YFLOC location is invalid for the currently displayed image. For example, if the image is 512 x 512 and you specify (700,700), an error message will be printed, unless /NO_WARN is set.

Use this program to move the current pixel of a given display. The Zoom window will also be moved, so that it is centered around the new current pixel location. If the new current pixel location is not within the current Image window, the Image and Scroll windows will also be moved, so that they include the new current pixel.

Calling Sequence

DISP_GOTO, DN, XFLOC, YFLOC, /NO_WARN
Arguments

DN

The display number corresponding to an open display. ENVI Display #1 will have DN = 0, Display #2 will have DN = 1, etc.

XFLOC

The x location of the new current pixel, in zero-based file coordinates.

YFLOC

The y location of the new current pixel, in zero-based file coordinates.

Keywords

NO_WARN

Set this keyword, if you do not want an error message printed when the XFLOC, YFLOC location is invalid for the currently displayed image. For example, if the image is 512 x 512 and you specify (700,700), an error message will be printed, unless /NO_WARN is set.

Example

```
; This example move the current pixel in
; the first display by ten pixels in X and Y.
;
pro example_disp_goto

; Get all the display numbers and
; check to make sure at least one
; display is present.
;
display = envi_get_display_numbers()
if (display[0] eq -1) then return
;
; Move the first display ten pixels
; in X and Y.
;
disp_get_location, display[0], xloc, yloc
disp_goto, display[0], xloc+10, yloc+10
end
```
See Also

DISP_GET_LOCATION, ENVI_DISP_QUERY
DISP_OUT_IMG

Use this program to output an image to a PostScript file.

**Calling Sequence**

DISP_OUT_IMG

**Keywords**

- **ANN_COLOR (optional)**
  Use this optional keyword to specify the grayscale level for graphical overlays when outputting to grayscale. ANN_COLOR is a single integer value from, 0 to 255 (inclusive). The default is zero.

- **ANN_NAMES (optional)**
  Use this optional keyword to specify an array of saved annotation file names. Each of these annotation files will be overlaid on the output image.

- **BORDER (optional)**
  Use this optional keyword to specify a two-element array of X and Y border sizes (in inches).

- **BPP (optional)**
  Use this optional keyword to specify the number of bits per pixel in the PostScript output. Valid values are 1, 2, 4, or 8.

- **COLOR**
  Set this keyword for color output, otherwise set to zero.

- **DIMS**
  Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:
  - DIMS(0): Unused for this function, set to -1.
  - DIMS(1): The starting X pixel. (The first pixel is number zero.)
  - DIMS(2): The ending X pixel.
• DIMS(3): The starting Y pixel. (The first pixel is number zero.)
• DIMS(4): The ending Y pixel.

**FID**

Use this keyword to specify an array of file IDs.

**LAND**

Set this keyword for landscape output, otherwise set to zero for portrait output.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting PostScript file.

**POS**

Use this keyword to specify an array of band positions for each of the files specified by the FID array.

**PSIZE**

Use this keyword to specify a two-element array indicating the paper size in inches (e.g. [8.5, 11]).

**REBIN**

Use this keyword to specify a REBIN factor to apply to the data. Values less than one enlarge the data, and values greater than one reduce the data.

**XOFF**

Use this keyword to specify the X offset (in inches) from the left side of the output.

**XSIZE**

Use this keyword to specify the X size of the output.

**YOFF**

Use this keyword to specify the Y offset (in inches) from the top of the output.

**YSIZE**

Use this keyword to specify the Y size of the output.
Example

pro example_disp_out_img
;
; First restore all the base save files.
; envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Set the DIMS keyword to process
; all spatial data and set the POS
; keyword to select the first band
; in the file.
;
envi_file_query, fid, ns=ns, nl=nl
dims = [-1, 0, ns-1, 0, nl-1]
pos = [0]
;
; Call the output routine.
;
disp_out_img, $  
    fid=fid, dims=dims, pos=pos, $  
    bpp=8, color=0, land=0, psize=[8.5,11], $  
    out_name='test.ps', rebin=1., $  
    xoff=.25, yoff=.25, xsize=5., ysize=5.
;
; Exit ENVI
;
envi_batch_exit
end
ELINE_CAL_DOIT

Use this program to spectrally calibrate an image using the Empirical Line method.

Calling Sequence

ENVI_DOIT, ‘ELINE_CAL_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.
PATH_RAD
Use this keyword to specify an array of Path Radiance values. There is one value for each band in the POS array.

POS
Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

SOLAR_IRR
Use this keyword to specify an array of Maximum Solar Irradiance values. There is one value for each band in the POS array.

Example

pro example_eline_cal_doit
;
; First restore all the base save files.
;
envi, /restore_base_save_files
;
; Initialize ENVI and send all errors and warnings to the file batch.txt
;
envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
   envi_batch_exit
   return
endif
;
; Set the keywords. We will perform the emperical line calibration on all samples and all bands in the file.
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
path_rad = [2.00, 1.33, 1.20, 1.11, 2.60, 3.12]
solar_irr = [2.33, 4.10, 6.00, 1.55, 5.32, 4.05]

; Perform the empirical line calibration
; envi_doit, 'eline_cal_doit', 
  fid=fid, pos=pos, dims=dims, 
  path_rad=path_rad, 
  solar_irr=solar_irr, 
  out_name=out_name, r_fid=r_fid 
;
; Exit ENVI
;
envi_batch_exit
end
EMITTANCE_CALC_DOIT

Use this program to convert data to emissivity, using either Reference Channel Emissivity, Emissivity Normalization, or Alpha Residuals. Using the keywords DATA_SCALE and WL_SCALE input data and wavelengths can be converted on the fly to $W/m^2/\mu m/sr$ and $\mu m$, respectively. Optionally, a temperature image can be output when using the Reference Channel Emissivity or Emissivity Normalization method.

Calling Sequence

ENVI_DOIT, ‘EMITTANCE_CALC_DOIT’

Keywords

CONSTANT_BAND (optional)

Use this keyword to specify the band position of the constant band used in the Reference Channel method. CONSTANT_BAND is a single long value ranging from zero to number of bands -1. CONSTANT_BAND is not used for the other methods.

DATA_SCALE

Use this keyword to specify the scale factor applied to the data, prior to calculating the emissivity. DATA_SCALE converts the input data to $W/m^2/\mu m/sr$.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

EMISSIVITY_VALUE (optional)

Use this keyword to specify the assumed emissivity value. EMISSIVITY_VALUE is only used for METHOD equal to 0 or 1.
**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**METHOD**

Set this keyword equal to one of the following values, to specify the algorithm for calculating the image data emissivity.

- 0 - Reference Channel
- 1 - Emissivity Normalization
- 2 - Alpha Residuals

**OUT_BNAME (optional)**

Use this keyword to specify a string array of output band names, if desired.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

**R_FID (optional)**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**TEMP_OUT_NAME (optional)**

Use this keyword to specify an output filename for the temperature image. If this item is present, the temperature image is automatically saved. TEMP_OUT_NAME is only valid for METHOD equal 0 or 1.
TEMP_IN_MEMORY (optional)

Set this keyword to specify that output temperature images should be stored in memory. TEMP_OUT_NAME is only valid for METHOD equal 0 or 1.

WL_SCALE

Use this keyword to specify the scale factor applied to the wavelengths, prior to calculating the emissivity. WL_SCALE converts the input wavelength to \textit{um}.

See Also

TIMS_CAL_DOIT
This procedure is used to restore the base ENVI save files for Batch mode. To restore the base save files, use the keyword RESTORE_BASE_SAVE_FILES.

Calling Sequence

ENVI

Keywords

RESTORE_BASE_SAVE_FILES

Use this keyword to restore the base ENVI save files. After calling this routine, batch processing can be initialized by ENVI_BATCH_INIT.

Example

This example restores the ENVI base save files and initializes ENVI for batch processing.

```
envi, /restore_base_save_files
openw, lunit, 'batch.log', /get_lun
en vi_batch_init, /batch_mode, batch_unit=lunit
```

See Also

ENVI_BATCH_INIT, ENVI_BATCH_EXIT
ENVI_ADD_PROJECTION

Use this procedure to update the ENVI projections.

Calling Sequence

ENVI_ADD_PROJECTION, Proj

Keywords

NAME (optional)

Use this optional keyword to specify the output file to write, including the directory path. The default is to use map_proj.txt in the map_proj directory of the installation tree. NAME is a string.

WRITE (optional)

Set this optional keyword to update the map projection file. The default is to add the projection to the current ENVI session and not save the projection to a file.
ENVI_AVHRR_CALIBRATE_DOIT

Use this program to calibrate AVHRR data or compute AVHRR Sea Surface Temperature (SST). The input to this routine must be an AVHRR level 1b or KLM file. The information in the file header is used for the calibration and SST calculations.

Calling Sequence

ENVI_DOIT, ‘ENVI_AVHRR_CALIBRATE_DOIT’

Keywords

CALC_SST (optional)

Set this keyword to output the Sea Surface Temperature (SST) instead of the calibrated data. If CALC_SST is set, you must also specify METHOD. The default is to not compute SST.

CORRECT_SOLARZ (optional)

Set this keyword to allow correction for the solar zenith angle. The default is to not correct for the solar zenith angle.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.
**IN_MEMORY (optional)**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**METHOD (optional)**

Use this keyword to specify the algorithm for the SST calculation. If METHOD is set you must also set CALC_SST. METHOD is an integer number with one of the following values:

- 0: Day MCSST Split
- 1: Night MCSST Split
- 2: Night MCSST Dual
- 3: Night MCSST Triple

**OUT_BNAME (optional)**

Use this keyword to specify a string array of output band names, if desired.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

**R_FID (optional)**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**Example**

```pro
pro example_envi_avhrr_calibrate_doit
;
; First restore all the base save files.
;
envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
```
; envi_batch_init, log_file='batch.txt'
;
; Open the data and class files
;
envi_open_data_file, 'avhrr_file', 
   r_fid=fid, /avhrr
if (fid eq -1) then begin
   envi_batch_exit
   return
endif
;
; Set the keywords
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1l, 0, ns-1, 0, nl-1]
out_name = 'testimg'
pos = lindgen(nb)
;
; Call the doit
;
envi_doit, 'envi_avhrr_calibrate_doit', 
   fid=fid, dims=dims, pos=pos, 
   out_name=out_name, /correct_solarz, 
   r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end

See Also

ENVI_AVHRR_GEOMETRY_DOIT
ENVI_AVHRR_GEOMETRY_DOIT

Use this program to compute the AVHRR geometry (latitude and longitude), solar zenith angles, and sensor zenith angles for each pixel. The input to this routine must be an AVHRR level 1b or KLM file. The information in the file header is used to compute the AVHRR geometry.

**Calling Sequence**

`ENVI_DOIT, ’ENVI_AVHRR_GEOMETRY_DOIT’`

**Keywords**

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**IN_MEMORY (optional)**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**METHOD**

Use this keyword to specify which of the output bands to calculate. METHOD is a four element array of ones and zeros, where a one indicates that the corresponding output band should be calculated. METHOD has the following definitions:

- METHOD[0] - Compute the latitude of each pixel
- METHOD[1] - Compute the longitude of each pixel
- METHOD[2] - Compute the solar zenith angle of each pixel
- METHOD[3] - Compute the sensor zenith angle of each pixel

**OUT_BNAME (optional)**

Use this keyword to specify a string array of output band names, if desired.

**OUT_DT**

Use this keyword to specify the output data type as either floating-point or double-precision. OUT_DT is a long integer that uses the IDL data type conventions (4=floating-point (32-bits), 5 = double-precision floating-point (64-bits)).

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**RADIANS (optional)**

Set this keyword to specify that the output zenith angles are in units of Radians. The default is to use units of Degrees.

**R_FID (optional)**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**Example**

```idl
pro example_envi_avhrr_geometry_doit
 ; ; First restore all the base save files.
 ; envi, /restore_base_save_files
 ; ; Initialize ENVI and send all errors
 ; and warnings to the file batch.txt
 ; envi_batch_init, log_file='batch.txt'
 ; ; Open the data and class files
 ; envi_open_data_file, 'avhrr_file', $
 ; r_fid=fid, /avhrr
 if (fid eq -1) then begin
```

```idl
```

---

**ENVI_AVHRR_GEOMETRY_DOIT**

*ENVI Programmer's Guide*
ENVI_Routines

ENVI Programmer’s Guide

ENVI_AVHRR_GEOMETRY_DOIT

environ_exit
return
d
; Set the keywords
;
environ_file_query, fid, ns=ns, nl=nl
dims = [-1l, 0, ns-1, 0, nl-1]
out_name = 'testimg'
method = [1,1,1,1]
out_bname = ['Latitude', $
    'Longitude', $
    'Solar Zenith', $
    'Sensor Zenith']
;
; Call the doit
;
environ_doit, 'environ_avhrr_geometry_doit', $
    fid=fid, dims=dims, out_name=out_name, $
    out_dt=4, method=method, $
    out_bname=out_bname, r_fid=r_fid
;
; Exit ENVI
;
environ_batch_exit
end

See Also

ENVI_AVHRR_CALIBRATE_DOIT
ENVI_AVHRR_WARP_DOIT

Use this program to warp AVHRR data or resulting AVHRR data products. This routine requires the FID of the file to warp and the corresponding AVHRR FID of the original data set. For example, you can warp a calibrated image cube or an AVHRR Sea Surface Temperature image by specifying the FID of the product and the FID of the original AVHRR data set. In order to warp a raw AVHRR data set, you supply the same file id. The original AVHRR data set must be an AVHRR level 1b or KLM file. The information in the file header is used to compute the warp points.

Calling Sequence

ENVI_DOIT, ‘ENVI_AVHRR_WARP_DOIT’

Keywords

AVHRR_FID

Use this keyword to specify the file ID for the original AVHRR data set, either an AVHRR Level 1b or KLM formatted file. This is the value returned from the keyword R_FID in the ENVI_OPEN_DATA_FILE procedure (make sure to use set the keyword AVHRR when you call ENVI_OPEN_DATA_FILE). FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

BACKGROUND

Use this keyword to specify the output image background value.

DATUM (optional)

Use this keyword to specify the datum for the map information projection. The default for Geographic is WGS-84 and the default for UTM and State Plane is North America 1927. All other projections default to no datum.

DEGREE

Use this keyword to specify the degree of the warp to perform.
**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- **DIMS(0)**: Unused for this function, set to -1.
- **DIMS(1)**: The starting X pixel. (The first pixel is number zero.)
- **DIMS(2)**: The ending X pixel.
- **DIMS(3)**: The starting Y pixel. (The first pixel is number zero.)
- **DIMS(4)**: The ending Y pixel.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**GCP_OUT_NAME (optional)**

Use this keyword to specify the output filename for the warp-points filename. GCP_OUT_NAME is a single string specifying an output filename.

**GRID**

Use this keyword to specify the number of points in the X and Y warping grid. GRID is a two element long array specifying the number of X and Y points (respectively) used to form an equally-spaced set of warp points.

**IN_MEMORY (optional)**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**METHOD**

Set this keyword to the warping method to use. METHOD is an integer set to one of the following:

- 0 - RST with nearest neighbor
- 1 - RST with bilinear
- 2 - RST with cubic convolution
- 3 - Polynomial with nearest neighbor
• 4 - Polynomial with bilinear
• 5 - Polynomial with cubic convolution
• 6 - Triangulation with nearest neighbor
• 7 - Triangulation with bilinear
• 8 - Triangulation with cubic convolution

**OUT_BNAME (optional)**

Use this keyword to specify a string array of output band names, if desired.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**PIXEL_SIZE**

Use this keyword to specify the X and Y pixel size. PIXEL_SIZE is a two-element double precision array of the output X and Y pixel sizes, respectively.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

**PROJ**

Use this keyword to specify the projection for the warped file. PROJ is a projection structure returned from ENVI_GET_PROJECTION or ENVI_PROJ_CREATE.

**R_FID (optional)**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**ZERO_EDGE (optional)**

Set this keyword to specify that the edges outside of any triangles be set to the value specified by BACKGROUND. The keyword is only used for Triangulation, METHOD 6, 7, or 8.
Example

forward_function envi_proj_units_translate, $
   envi_proj_create

pro example_envi_avhrr_warp_doit
   ; First restore all the base save files.
   ; envi, /restore_base_save_files
   ; Initialize ENVI and send all errors
   ; and warnings to the file batch.txt
   ; envi_batch_init, log_file='batch.txt'
   ; Open the data and class files
   ; envi_open_data_file, 'avhrr_file', $
      r_fid=fid, /avhrr
   if (fid eq -1) then begin
      envi_batch_exit
      return
   endif
   ; Set the keywords
   ; envi_file_query, fid, ns=ns, nl=nl, nb=nb
   dims = [-1l, 0, ns-1, 0, nl-1]
   pos = lindgen(nb)
   out_name = 'testimg'
   units = envi_proj_units_translate('Degrees')
   proj = envi_proj_create(/geographic, unit=units)
   ps = [.05,.05]
   ; Call the doit
   ; envi_doit, 'envi_avhrr_warp_doit', fid=fid, $
      avhrr_fid=fid, dims=dims, pos=pos, $
      out_name=out_name, method=1, degree=1, $
      background=0, grid=[5,5], proj=proj, $
      pixel_size=ps, r_fid=r_fid
   ; Exit ENVI
   ; envi_batch_exit
end
See Also

ENVI_AVHRR_GEOMETRY_DOIT
Chapter 9: ENVI Routines

ENVI Programmer

ENVI_BATCH_EXIT

Use this new routine to exit ENVI from the non-menu “batch” mode, which is the mode without the ENVI menu interface. This routine will conclude the ENVI session. If the ENVI preference “Exit IDL on Exit from ENVI” is set to “Yes”, then the IDL session will also terminate. (To set ENVI preferences, see the ENVI User’s Guide or online help.) ENVI_BATCH_EXIT closes the session that was started with ENVI_BATCH_INIT.

Calling Sequence

ENVI_BATCH_EXIT

Example

The following procedure and example code will initialize ENVI in batch mode, set the log file to batch.txt, print “Hello World” and exit ENVI:

1. In the procedure use the ENVI command with the keyword RESTORE_BASE_SAVE_FILES to restore the base ENVI save files.
2. Initialize ENVI in the non-menu batch mode using ENVI_BATCH_INIT.
3. Use the IDL print command to print “Hello World”.
4. Exit ENVI using the ENVI_BATCH_EXIT command.

To execute the example routine:

1. Start IDL without starting ENVI.
2. Compile and execute the procedure.

```idl
pro example_envi_batch_exit

; First restore all the base save files.
; envi, /restore_base_save_files

; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'

; Print message
; print, 'Hello World'
```
; Exit ENVI

; envi_batch_exit
end

See Also

ENVI, ENVI_BATCH_INIT, ENVI_DOIT
ENVI_BATCH_INIT

Use this routine to initialize ENVI in the non-menu “batch” mode, which is the mode without the ENVI menu interface. Many of the ENVI image processing programs that do not require user interaction were designed to be run in non-menu batch mode.

Warning

Do not attempt to initialize ENVI in batch mode from and IDL session that is currently running an interactive ENVI session. Instead, start a new IDL session to initialize ENVI in batch mode.

Calling Sequence

ENVI_BATCH_INIT

Keywords

LOG_FILE (optional)

Use this keyword to specify a batch log file. Any errors or warnings that would normally appear on the screen during an interactive ENVI session will be written to the log file. It is important to check your log file after your batch processing routine is finished, to make sure no errors have occurred. LOG_FILE is a string variable specifying the filename and path of the batch log file.

NO_STATUS_WINDOW (optional)

Set this optional keyword to prevent display of the ENVI processing status window. Additional control of the status window can be achieved using ENVI_BATCH_STATUS_WINDOW.

Example

See the example under ENVI_BATCH_EXIT.

See Also

ENVI, ENVI_BATCH_EXIT, ENVI_BATCH_STATUS_WINDOW, ENVI.DOIT
ENVI_BATC H_STA TUS_WINDOW

Use this routine to enable and disable the ENVI batch status window. The status window is used to show the progress of the processing routines. The keywords ON and OFF control whether the window is enabled (ON) or disabled (OFF). ENVI_BATC H_STA TUS_WINDOW allows the user precise control of the status window and may be used to enable and disable the status window multiple times in a single batch processing session. In addition, the keyword NO_STATUS_WINDOW to the ENVI_BATC H_INIT routine controls the initial state of the status window.

Calling Sequence

ENVI_BATC H_STA TUS_WINDOW

Keywords

OFF (optional)

Set this optional keyword to prevent display of the ENVI processing status window.

ON (optional)

Set this optional keyword to allow display of the ENVI processing status window.

Example

The following example will run statistics on bhtmref.img with and without the status window. First, ENVI is initialized in batch mode with the status window present. The file bhtmref.img is opened and ENVI_STATS_DOIT is used to perform basic statistical calculations. During this process, the status window will be displayed. Next, the status window is turned off and the same statistics processing is performed. This time, the status window is not displayed. After each statistical calculation the mean value is printed.

This example uses the following files found in the data directory of the ENVI installation:

- bhtmref.img
- bhtmref.hdr

```
pro example_envi_batch_status_window

; First restore all the base save files.
```
envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
   envi_batch_exit
   return
endif
;
; Get the samples, lines and # bands
; for the input file.
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb
;
; Set the dims and pos to calculate
; statistics for all data (spatially and
; spectrally) in the file.
;
dims = [-1l, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
;
; Calculate the basic statistics and
; print the mean.
;
envi_doit, 'environ_stats_doit', fid=fid, pos=pos, $
   dims=dims, comp_flag=1, mean=mean
print, 'Mean', mean
;
; Turn off the status window and do the same
;
envi_batch_status_window, /off
;
; Calculate the basic statistics and
; print the mean.
;
envi_doit, 'environ_stats_doit', fid=fid, pos=pos, $
   dims=dims, comp_flag=1, mean=mean
print, 'Mean', mean
;
; Exit ENVI
;
envi_batch_exit
end
See Also

ENVI_BATCH_INIT, ENVI_DOIT
ENVI_BUFFER_ZONE_DOIT

Use this program to create a buffer-zone image from a classification image. Each pixel in the output image is the nearest distance, in pixels, from any classified pixel specified by CLASS_PTR. Any pixels which are farther away from the MAX_DISTANCE are set to MAX_DISTANCE.

Calling Sequence

ENVI_DOIT, ‘ENVI_BUFFER_ZONE_DOIT’

Keywords

CLASS_PTR

Use this keyword to specify the classes around which to compute the buffer zone. CLASS_PTR is a long array of class numbers ranging from zero (Unclassified) to the number of classes.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

DISTANCE_DT (optional)

Use this keyword to specify the output data type. DISTANCE_DT is an integer value set to zero or one indicating either integer or floating point distance, respectively. Depending on the maximum distance, integer data types are either byte, unsigned integer, or unsigned long integer. The default is one (floating point output).
FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY (optional)

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

MAX_DISTANCE

Use this keyword to specify the maximum distance for the buffer zone. MAX_DISTANCE is a long integer in pixels specifying the maximum buffer zone to compute. Pixels greater than MAX_DISTANCE will be set to MAX_DISTANCE.

OUT_BNAME (optional)

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID (optional)

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

pro example_envi_buffer_zone_doit
;
; First restore all the base save files.
;
envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
Chapter 9: ENVI Routines

ENVI Programmer's Guide

ENVI_BUFFER_ZONE_DOIT

; envi_batch_init, log_file='batch.txt'
; Open the input file
; envi_open_file, 'bhtm_sam.img', r_fid=fid
if (fid eq -1) then begin
   envi_batch_exit
   return
endif
;
; Set the POS and DIMS keywords to processes the first band (classification images only have one band) and all the pixels. The MAX_DISTANCE and CLASS_PTR keywords are set to create a buffer zone of 20 pixels around the second class (remember the first class is the unclassified class).
;
envi_file_query, fid, ns=ns, nl=nl
pos = [0]
dims = [-1l, 0, ns-1, 0, nl-1]
out_name = 'testimg'
max_distance = 20
distance_dt = 1
class_ptr = [1]
;
; Call the doit
;
envi_doit,'envi_buffer_zone_doit', $
   fid=fid, pos=pos, dims=dims, $
   max_distance=max_distance, class_ptr=class_ptr, $
   distance_dt=distance_dt, out_name=out_name
;
; Exit ENVI
;
envi_batch_exit
end
ENVI_CAL_DOIT

Use this program to spectrally calibrate an image using Flat Field or IARR.

Calling Sequence

ENVI_DOIT, 'ENVI_CAL_DOIT'

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

DSTR

Use this keyword to specify a named variable that will hold one of the following descriptions (depending on which method was used to calculate the mean values): 'Flat Field' or 'IARR Calibration'.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

MEAN

Use this keyword to specify an array of values to divide into each band. MEAN is an array of floating-point numbers equal in size to the number of bands specified.
OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```pro
pro example_envi_cal_doit
    ; First restore all the base save files.
    ; envi, /restore_base_save_files
    ; Initialize ENVI and send all errors and warnings to the file batch.txt
    ; envi_batch_init, log_file='batch.txt'
    ; Open the input file
    ; envi_open_file, 'bhtmref.img', r_fid=fid
    if (fid eq -1) then begin
        envi_batch_exit
        return
    endif
    ; Set the keywords. We will perform the flat field calibration on all samples and all bands in the file.
    ; envi_file_query, fid, ns=ns, nl=nl, nb=nb
    dims = [-1, 0, ns-1, 0, nl-1]
    pos = lindgen(nb)
```
out_name = 'testimg'
mean = [1.39, 2.15, 3.41, 1.73, 1.92, 4.32]
dstr = 'Flat Field'

; Perform the Flat Field calibration
; envi_doit, 'envi_cal_doit', $
  fid=fid, pos=pos, dims=dims, $
  mean=mean, dstr=dstr, $
  out_name=out_name, r_fid=r_fid

; Exit ENVI
; envi_batch_exit
end
ENVI_CENTER

Use this procedure to get the centering offsets for a widget.

Calling Sequence

ENVI_CENTER, Xoff, Yoff

Arguments

XOFF

A named variable that will contain the X (horizontal) offset for use in the call to
WIDGET_BASE.

YOFF

A named variable that will contain the Y (vertical) offset for use in the call to
WIDGET_BASE.

Keywords

XBIG (optional)

Set this optional keyword if the widget will be large in the horizontal direction.

YBIG (optional)

Set this optional keyword if the widget will be large in the vertical direction.

Example

This example gets the centering information from ENVI_CENTER and uses it in the
call to WIDGET_BASE.

```plaintext
  envi_center, xoff, yoff
  base = widget_base(title='Center test', xoff=xoff, yoff=yoff, /column)
```
ENVI_CLOVER_DOIT

Use this program to overlay a classification image on the output image.

Calling Sequence

ENVI_DOIT, ‘ENVI_CLOVER_DOIT’

Keywords

CFID

Use this keyword to specify the file ID for the classification image. This is the value returned from the keyword F_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
OVERLAY_LUT

Use this keyword to specify an array indicating which classes to overlay on the image. For example, to overlay classes 2, 3 and 4, set the keyword as follows:

\[
\text{overlay\_lut} = [2, 3, 4]
\]

Note: class numbers are zero based and the unclassified class is usually class zero.

If you overlay all the classes on the output image, the output image will not be visible. Typically, unclassified pixels are not overlain.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```pro
pro example_clover_doit
 ; ; First restore all the base save files.
 ; envi, \restore_base_save\_files
 ; ; Initialize ENVI and send all errors
 ; and warnings to the file batch\_txt
 ; envi\_batch\_init, log\_file='batch\_txt'
 ; ; Open the data and class files
 ; envi\_open\_file, 'bhtmref\_img', r\_fid=fid
 envi\_open\_file, 'bhtm\_sam\_img', r\_fid=cfid
```
if (fid eq -1 or cfid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Set the necessary variables
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = [0,1,2]
out_name = 'testimg'
overlay_lut = [1,4,5]
cpos = [0]
;
; Call the doit
;
envi_doit, 'envi_clover_doit', $
    fid=fid, pos=pos, dims=dims, $
    cfid=cfid, cpos=cpos, dn=-1, $
    overlay_lut=overlay_lut, $
    out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
ENVI_COLLECT_SPECTRA

Use this routine to perform spectra collection. Spectra collection is the collection of spectral signatures directly from ASCII files, spectral libraries, ROIs, statistics files, or by using the “drag-and-drop” method. The ENVI_COLLECT_SPECTRA dialog is not modalized, which allows the user to interact with ENVI and collect spectra using any of the above techniques. The collected spectra are often used in spectral-based processing such as unmixing or classification.

The collected spectra are provided to the user through the procedure defined by the keyword PROCEDURE. After at least one spectrum has been collected, selecting the “Apply” button will call the defined procedure. The procedure call provides information supplied by the user (keywords: FID, DIMS, POS, M_FID, M_POS, INFO) plus information about the collected spectra (spectral data, names, and color).

Note

For more information, see “Using the Endmember Collection Widget” on page 182.

Calling Sequence

ENVI_COLLECT_SPECTRA

Keywords

DIMS (optional)

Use this optional keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-elements array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

The default is to use DIMS = [-1, 0, ns-1, 0, nl-1].
**FID**

Use this keyword to specify the file ID for the open file. This is the value returned by the R_FID keyword to the ENVI_OPEN_FILE procedure, or the FID keyword to ENVI_SELECT. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1. The value of FID is passed to PROCEDURE with the FID keyword.

**INFO (optional)**

Use this optional keyword to specify any user-defined data to pass to the routine defined by PROCEDURE. The value of INFO can be any IDL variable, including structures. The value of INFO is passed to PROCEDURE with the INFO keyword.

**M_FID (optional)**

Use this optional keyword to specify the file ID for the mask file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure, or the FID keyword to ENVI_SELECT. M_FID is a long integer with a value greater than zero. An invalid file ID is specified as -1. The value of M_FID is passed to PROCEDURE with the M_FID keyword.

**M_POS (optional)**

Use this optional keyword to specify the band position of the mask band. M_POS is a single long value greater than or equal to zero and less than the number of bands in the file specified by M_FID. The value of M_POS is passed to PROCEDURE with the M_POS keyword.

**POS (optional)**

Use this optional keyword to specify an array of band positions. POS is an array of long integers that range from zero to (number of bands)-1. The default is POS=LINDGEN(NB), where the number of bands is retrieved from the file specified by FID. The value of POS is passed to PROCEDURE with the POS keyword.

**PROCEDURE**

Use this keyword to specify the procedure to call when the “Apply” button is selected. PROCEDURE must have the following definition even if you do not specify the optional keywords to ENVI_COLLECT_SPECTRA:

```idl
PRO MY_PROCEDURE, FID=FID, POS=POS, DIMS=DIMS, $
   INFO=INFO, M_FID=M_FID, M_POS=M_POS, SPEC=SPEC, $
   SNAMES=SNAMES, SCOLORS=SCOLORS, _EXTRA=_EXTRA
```
Where FID, POS, DIMS, INFO, M_FID and M_POS are passed through from the call to ENVI_COLLECT_SPECTRA. SPEC, SNAMES and SCOLORS contain information about the collected spectra. The SPEC array contains the data for the collected spectra \[nb, nspec\]; SNAMES is a string array of the collected spectra names; SCOLOR is an array of graphics color indexes. To convert graphics colors to RGB triplets, see ENVI_GET_RGB_TRIPLETS. The _EXTRA keyword is used to collect unused keywords.

**TITLE (optional)**

Use this optional keyword to specify the string used in the title bar of the Spectral Collection dialog window. Enter the title enclosed in single quotes. The default is “Endmember Collection”

**Example**

```plaintext
pro example_envi_collect_spectra
  envi_select, fid=fid
  if (fid eq -1) then return
  envi_collect_spectra, fid=fid, procedure='my_procedure'
end

; Define the procedure to called each time the apply button is selected.
; pro my_procedure, fid=fid, pos=pos, dims=dims, spec=spec, $ snames=snames, scolors=scolors, _extra=_extra
  print, 'snames ', snames
  print, 'scolors ', scolors
  print, 'spec ', spec
end
```
ENVI_COMPUTE_SUN_ANGLES

Use this function to compute the Sun elevation and azimuth angles for a given time, latitude, and longitude. This function returns a two element double array of the Sun’s elevation and azimuth for the selected input values.

Calling Sequence

\[
\text{Result} = \text{ENVI\_COMPUTE\_SUN\_ANGLES} \\
(\text{Day, Month, Year, GMTtime, Lat, Lon})
\]

Arguments

Day

Use this keyword to specify the day of the year for the desired sun angles. Day is an integer value between and including 1 to 31.

GMTtime

Use this keyword to specify the GMT time for the desired sun angles. GMTtime is floating point value between and including 0 to 2400. For example, set GMTtime equal to 1450. to represent a GMT time of 2:30pm.

Lat

Use this keyword to specify the latitude for the desired sun angles. Lat is a floating point number in degrees. North is positive and South negative.

Lon

Use this keyword to specify the longitude for the desired sun angles. Lon is a floating point number in degrees. East is positive and West is negative.

Month

Use this keyword to specify the month of the year for the desired sun angles. Month is an integer from 1 to 12.

Year

Use this keyword to specify the year for the desired sun angles. Year is a four-digit integer value of the form yyyy.
Example

This example computes the Sun’s elevation and azimuth angles for N34, W117 July 4, 1984 GMT 10:45pm.

```
angles = envi_compute_sun_angles(4, 7, 1984, 2275., $ 34., -117.)
print, 'Sun elevation= ', angles[0]
print, 'Sun azimuth= ', angles[1]
```

See Also

TOPO_DOIT
ENVI_CONVERT_FILE_COORDINATES

Use this procedure to convert between map and pixel, coordinated for a particular file. This utility will convert X and Y pixel coordinates into their corresponding X and Y map coordinates and vice-versa. The FID argument is used to retrieve the appropriate map projection information.

Calling Sequence

ENVI_CONVERT_FILE_COORDINATES, Fid, XF, YF, XMap, YMap

Arguments

FID

Use this keyword to specify a named variable that will contain the file ID for the open file. The file ID is used to get the appropriate projection information for the conversion. If the file does not have an associated projection, the file coordinates and map coordinates are equal. FID is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. An invalid file ID is specified as -1.

XF

When the keyword TO_MAP is set, XF is a variable containing the X file coordinates to convert. XF can be a single value or an array of values. The first file coordinate is zero.

When the keyword TO_MAP is not set, XF is a named variable that will contain the returned X file coordinates for the input XMap and YMap arrays.

YF

When the keyword TO_MAP is set, YF is a variable containing the Y file coordinates to convert. YF can be a single value or an array of values. The first file coordinate is zero.

When the keyword TO_MAP is not set, YF is a named variable that will contain the returned Y file coordinates for the input XMap and YMap arrays.
XMap

When the keyword TO_MAP is set, XMap is a named variable that will contain the returned X map coordinates for the input XF and YMap arrays.

When the keyword TO_MAP is not set, XMap is a variable containing the X map coordinates to convert. XMap can be a single value or an array of values.

YMap

When the keyword TO_MAP is set, YMap is a named variable that will contain the returned Y map coordinates for the input XF and YF arrays.

When the keyword TO_MAP is not set, YMap is a variable containing the Y map coordinates to convert. YMap can be a single value or an array of values.

Keywords

TO_MAP

Set this keyword to convert the input file coordinates to map coordinates.

Example

This example converts the first three pixels on the first line to map coordinates.

```
xf = [0,1,2]
yf = [0,0,0]
envi_convert_file_coordinates, fid, xf, yf, xmap, ymap, /to_map
```

See Also

WIDGET_GEO, WIDGET_MAP
ENVI_CONVERT_FILE_MAP_PROJECTION

Use this routine to convert a file from its current map projection to the specified output projection. This routine requires an input file, the output projection, and the resampling and warping method; no ground control points are needed (they are generated internally). This routine will convert among any projections in ENVI. Neither the input or output projection can be Arbitrary.

Calling Sequence

ENVI_CONVERT_FILE_MAP_PROJECTION

Keywords

BACKGROUND (optional)

Use this optional keyword to specify the output image background value. All pixels outside the warped-image boundary will be set to the value specified by BACKGROUND. The default is zero.

DEGREE (optional)

Use this optional keyword to specify the degree of the warping polynomial for the Polynomial method. The degree of the polynomial is limited by the number of GCPs. You must ensure that \#GCPs > (degree + 1)^2. The Default is one (first degree).

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. The input file must be georeferenced in any projection other than Arbitrary. FID is the value returned from
the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**GCP_NAME (optional)**

Use this keyword to specify the output filename for the warp-points filename. GCP_NAME is a single string specifying an output filename.

**GRID (optional)**

Use this keyword to specify the grid spacing in pixels for the X and Y warp points. GRID is a two-element long array, specifying the X and Y grid spacing (respectively) in pixels for the warp points used to convert between the two map projections. Regardless of the GRID value, a minimum of the four corner points will be used. The default is to use every 10th point in both X and Y.

**O_PIXEL_SIZE**

Use this keyword to specify the X and Y pixel size. O_PIXEL_SIZE is a two-element double-precision array of the output X and Y pixel sizes, respectively.

**O_PROJ**

Use this keyword to specify the output projection for the converted file. O_PROJ cannot be set to the Arbitrary projection. O_PROJ is a projection structure returned from ENVI_GET_PROJECTION or ENVI_PROJ_CREATE.

**OUT_BNAME (optional)**

Use this optional keyword to specify a string array of output band names, if desired.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. OUT_NAME is a string variable specifying the filename and path of the output file.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

**R_FID (optional)**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.
RESAMPLING (optional)
Set this keyword equal to one of the following values to specify the resampling method:

- 0 - Nearest Neighbor
- 1 - Bilinear
- 2 - Cubic Convolution

The default is to use Nearest Neighbor, which is method “0”.

WARP_METHOD (optional)
Set this optional keyword equal to one of the following values to specify the warp method:

- 0 – RST
- 1 – Nth Degree Polynomial
- 2 - Triangulation

The default is to use RST, which is method “0”.

ZERO_EDGE (optional)
Set this optional keyword to specify that the edges outside of any triangles be set to the value specified by BACKGROUND. The keyword is used only for Triangulation (method “2”).

Example
The following example will convert the file bhtmref.img from UTM zone 13 North to UTM zone 12 North. The output UTM projection is created using ENVI_PROJ_CREATE, with UTM zone 12 North, the default datum of North America 1927, and the default units of Meters. The output pixel size is set to 28.5 meters for both X and Y. The conversion will use the RST method with bilinear resampling.

This example uses the following files found in the DATA directory of the ENVI installation:

- bhtmref.img
- bhtmref.hdr

forward_function envi_proj_create
pro envi_convert_file_map_projection
  ; First restore all the base save files.
  ;
  envi, /restore_base_save_files
  ; Initialize ENVI and send all errors
  ; and warnings to the file batch.txt
  ;
  envi_batch_init, log_file='batch.txt'
  
  ; Open the input file
  ;
  ; Open the input file
  ;
  envi_open_file, 'bhtmref.img', r_fid=fid
  if (fid eq -1) then begin
    envi_batch_exit
    return
  endif
  
  ; Setup the values for the keywords
  ;
  ;
  envi_file_query, fid, ns=ns, nl=nl, nb=nb
  pos = lindgen(nb)
  dims = [-1l, 0, ns-1, 0, nl-1]
  out_name = 'testimg'
  o_proj = envi_proj_create(/utm, zone=12)
  o_pixel_size = [28.5, 28.5]
  
  ; Call the doit
  
  envi_convert_file_map_projection, fid=fid, $
  pos=pos, dims=dims, o_proj=o_proj, $
  o_pixel_size=o_pixel_size, grid=[50,50], $
  out_name=out_name, warp_method=0, $
  resampling=1, background=0
  ;
  ; Exit ENVI
  ;
  envi_batch_exit
end

See Also

ENVI_CONVERT_FILE_COORDINATES, ENVI_CONVERT_PROJECTION_COORDINATES, REG_WARP_DOIT
ENVI_CONVERT_PROJECTION_COORDINATES

Use this routine to convert map coordinates from one projection to another. The routine supports conversion among any ENVI projection types including user-defined. For example, this routine can convert UTM coordinates to Geographic (Latitude and Longitude) or between UTM zones.

Calling Sequence

ENVI_CONVERT_PROJECTION_COORDINATES, iXmap, iYmap, iProj, oXmap, oYmap, oProj

Arguments

iProj

Use this argument to specify the input projection for the iXmap and iYmap points. iProj is a projection structure returned from ENVI_GET_PROJECTION or ENVI_PROJ_CREATE.

iXmap

Use this argument to specify the input X map points which will be converted from the iProj projection to the oProj projection. iXmap can be a single value or an array of values.

iYmap

Use this argument to specify the input Y map points, which will be converted from the iProj projection to the oProj projection. iYmap can be a single value, or an array of values.

oProj

Use this argument to specify the output projection for the converted iXmap and iYmap points. oProj is a projection structure returned from ENVI_GET_PROJECTION or ENVI_PROJ_CREATE.
**oXmap**

Use this argument to specify a named variable that will contain the converted X map points. The oXmap points are the input X map coordinates converted to the oProj projection.

**oYmap**

Use this argument to specify a named variable that will contain the converted Y map points. The oYmap points are the input Y map coordinates converted to the oProj projection.

**Keywords**

**INPUT_Z (optional)**

Use this keyword to specify the input elevation values corresponding to each of the input iXmap and iYmap points. Elevation values are used to more accurately convert between different data. INPUT_Z is a floating point array having the same length as iXmap.

**OUTPUT_Z (optional)**

Use this optional keyword to specify a named variable that will contain the output elevation values for each of the converted map coordinates. OUTPUT_Z is set only if INPUT_Z is specified.

**Example**

This example converts a series of latitude and longitude points in a Geographic projection to a UTM Zone 11 projection. The new map points are converted to a UTM zone 12 projection. First, we will create the Geographic and UTM zone 11 projections using ENVI_PROJ_CREATE. Next, we create an array of XY latitude and longitude map coordinates. These are the points that will be converted to the UTM zone 11 projection using ENVI_CONVERT_PROJECTION_COORDINATES. The results of the UTM zone 11 map coordinates will not be converted to UTM zone 12 map coordinates.

```plaintext
; ; Build the projections using
; the standard defaults
;
iproj = ENVI_PROJ_CREATE(/geographic)
op proj = ENVI_PROJ_CREATE(/utm,zone=11)
;
; Create the pairs of Latitude and Longitude
```
ixmap = [-117., -117.5, -118., $ -117., -117.5, -118.]
iymap = [34.0, 34.0, 34.0, $ 35.0, 35.0, 35.0]

; Convert from Geographic to UTM zone 11
ENVI_CONVERT_PROJECTION_COORDINATES, $ ixmap, iymap, iproj, $ oxmap, oymap, oproj

; Convert from UTM zone 11 to UTM zone 12
oproj2 = ENVI_PROJ_CREATE (/utm,zone=12)
ENVI_CONVERT_PROJECTION_COORDINATES, $ oxmap, oymap, oproj, $ oproj2, oymap2, oproj2

See Also

ENVI_GET_PROJECTION, ENVI_PROJ_CREATE, ENVI_MAP_INFO_CREATE
ENVI_CREATE_ROI

This function is used to create a new Region of Interest (ROI). ENVI_CREATE_ROI returns the ROI id of the newly created region. Points are added to the ROI using the routine ENVI_DEFINE_ROI (see “ENVI_DEFINE_ROI” on page 376).

Calling Sequence

Result = ENVI_CREATE_ROI()

Keywords

COLOR (optional)
Use this keyword to specify the graphics color index of the ROI. The default graphics color index is 2.

NAME (optional)
Use this keyword to specify a string variable for the ROI name. The default for NAME is 'New Region'.

NL
Use this keyword to specify the ROI spatial-line dimension tied to the ROI.

NS
Use this keyword to specify the ROI spatial sample dimension tied to the ROI.

Example

Use ENVI_CREATE_ROI to create a new ROI. Then, add a polygon to the ROI using ENVI_DEFINE_ROI.

    roi_id = envi_create_roi(color=4, name='Test roi', ns=512, nl=512)
    xpts = [100, 200, 200, 100, 100]
    ypts = [100, 100, 200, 200, 100]
    envi_define_roi, roi_id, /polygon, xpts=xpts, ypts=ypts

See Also

ENVI_GET_ROI_IDS, ENVI_GET_ROI_DIMS_PTR, ENVI_GET_ROI_DATA, ENVI_RESTORE_ROIS, ENVI_DEFINE_ROI
ENVI_CUBE_3D_DOIT

Use this program to build a 3D image cube.

**Calling Sequence**

ENVI_DOIT, ‘ENVI_CUBE_3D_DOIT’

**Keywords**

**BORDER**

Use this keyword to specify the number of border pixels surrounding the image cube.

**CT**

Use this keyword to specify the color table used for the spectral-profile colors. CT is a byte array (256, 3).

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- **DIMS(0):** Unused for this function, set to -1.
- **DIMS(1):** The starting X pixel. (The first pixel is number zero.)
- **DIMS(2):** The ending X pixel.
- **DIMS(3):** The starting Y pixel. (The first pixel is number zero.)
- **DIMS(4):** The ending Y pixel.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify a one-dimensional array of band positions for the spectral profile. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

RGB_POS
Use this keyword to specify the three RGB band positions for the cube face.

SCALE
Use this keyword to specify a spectral scale factor that can be used to exaggerate or compress the spectral profile. A value greater than 1.0 exaggerates the spectral profile; a value less than 1.0 compresses the spectral profile. SCALE must be set to 1.0, if the spectral profile is not to be changed.

Example

```
pro example_envi_cube_3d_doit
    ; ; First restore all the base save files.
    ; envi, /restore_base_save_files
    ; ; Initialize ENVI and send all errors
    ; and warnings to the file batch.txt
    ; envi_batch_init, log_file='batch.txt'
    ; ; Open the input file
    ; envi_open_file, 'bhtmref.img', r_fid=fid
    if (fid eq -1) then begin
        envi_batch_exit
        return
    endif
    ;
```
; Set the keywords DIMS and POS to
; use all pixels and all bands for the
; spatial and spectral parts of the cube.
; Use band 3,2,1 as the RGB bands for the
; front face of the cube.
; envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1L, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
rgb_pos = [3,2,1]
out_name = 'testimg'
;
; Set the CT keyword to the fifth color
; table from the IDL colors1.tbl file.
; openr, unit, filepath('colors1.tbl', $    sub=['resource','colors']), $ /block, /get
a = assoc(unit, bytarr(256,3), 1)
ct = a[4]
ct[0,*] = 0
free_lun, unit
;
; Call the doit
;
envi_doit, 'envi_cube_3d_doit', $    fid=fid, pos=pos, dims=dims, $    out_name=out_name, scale=10.0, $    border=30, rgb_pos=rgb_pos, ct=ct, $ r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
ENVI_DEFAULT_STRETCH_CREATE

Use this routine to return an ENVI default stretch structure. The default stretch is an element of the ENVI header and is used to set the default stretch when displaying data from a file. To set a default stretch, create the default stretch structure using ENVI_DEFAULT_STRETCH_CREATE and set the DEF_STRETCH keyword to the ENVI_SETUP_HEAD or ENVI_ENTER_DATA routines. The keywords LINEAR, PCT_LINEAR, GAUSSIAN, EQUALIZE, SQUARE_ROOT and NONE define the type of stretch created. The default type is NONE. The VAL1 and VAL2 keywords are used to specify required values for defining the different stretch types.

This routine should be used instead of directly accessing the default stretch structure.

Calling Sequence

Result = ENVI_DEFAULT_STRETCH_CREATE()

Keywords

EQUALIZE (optional)

Set this optional keyword to specify that an Equalization default stretch structure is created. The keywords VAL1 and VAL2 are not used with this stretch.

GAUSSIAN (optional)

Set this optional keyword to specify that a Gaussian default stretch structure is created. Use the keyword VAL1 to specify the number of gaussian standard deviations for the stretch. The default for VAL1 is 2. The keyword VAL2 is not used with this stretch.

LINEAR (optional)

Set this optional keyword to specify that a Linear Range default stretch structure is created. Use the keyword VAL1 to specify the minimum and VAL2 to specify the maximum values for the linear stretch range. The defaults for VAL1 and VAL2 are 0 and 255, respectively.

NONE (optional)

Set this optional keyword to create default stretch structure with no stretch defined. This is the default stretch created if no stretch type is specified. The keywords VAL1 and VAL2 are not used with this stretch.
PCT_LINEAR (optional)

Set this optional keyword to specify that a Percent Linear default stretch structure is created. Use the keyword VAL1 to specify the percentage for the stretch. For example, for a 2% stretch set VAL1=2. The default for VAL1 is zero, specifying a 0% stretch.

SQUARE_ROOT (optional)

Set this optional keyword to specify that a Square Root default stretch structure is created. The keywords VAL1 and VAL2 are not used with this stretch.

VAL1 (optional)

Use this optional keyword to specify one of the following when the indicated STRETCH_TYPE keyword is set:

- EQUALIZE: VAL1 is not used.
- GAUSSIAN: VAL1 is a floating-point number specifying the gaussian standard deviation for the stretch
- LINEAR: VAL1 is a floating-point number specifying the minimum value for the stretch range
- NONE: VAL1 is not used.
- PCT_LINEAR: VAL1 is a long integer from 0 to 100 (inclusive) specifying the percentage stretch.
- SQUARE_ROOT: VAL1 is not used.

VAL2 (optional)

Use this optional keyword to specify one of the following when the indicated STRETCH_TYPE keyword is set:

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQUALIZE</td>
<td>VAL2 is not used.</td>
</tr>
<tr>
<td>GAUSSIAN</td>
<td>VAL2 is not used.</td>
</tr>
<tr>
<td>LINEAR</td>
<td>VAL1 is a floating-point number specifying the maximum value for the stretch range</td>
</tr>
</tbody>
</table>

Table 9-2: VAL2 Keyword Values
Example

The following example will create a default stretch structure for a 2% Linear stretch

```python
def_stretch = ENVI_DEFAULT_STRETCH_CREATE(/PCT_LINEAR, VAL1=2)
```

The following example will create a default stretch structure for a Linear Stretch between 100 and 200.

```python
def_stretch = ENVI_DEFAULT_STRETCH_CREATE(/LINEAR, VAL1=100, VAL2=200)
```

See Also

ENVI_ENTER_DATA, ENVI_SETUP_HEAD

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>VAL2 is not used.</td>
</tr>
<tr>
<td>PCT_LINEAR</td>
<td>VAL2 is not used.</td>
</tr>
<tr>
<td>SQUARE_ROOT</td>
<td>VAL2 is not used.</td>
</tr>
</tbody>
</table>

*Table 9-2: VAL2 Keyword Values (Continued)*
ENVI_DEFINE_ROI

Use this procedure to define objects in a Region of Interest (ROI). An ROI is defined by point, polyline, and polygon objects. Each ROI can be composed of multiple and/or mixed objects. For example, a single ROI may have 100 individual points and three polygons. Using a ROI_ID returned from ENVI_CREATE_ROI, each call to ENVI_DEFINE_ROI will add one object to the ROI definition (multiple points may be added with a single call). Once an ROI is created, it can be used in processing routines, or saved to an ROI file.

Calling Sequence

ENVI_DEFINE_ROI, Roi_id

Arguments

ROI_id

Roi_id is a single ID returned from the function ENVI_CREATE_ROI or ENVI_GET_ROI_IDS.

Keywords

POINT (optional)

Set this optional keyword to indicate that the X and Y arrays define a set of points. When POINT is set, POLYLINE and POLYGON cannot be set.

POLYLINE (optional)

Set this optional keyword to indicate that the X and Y arrays define a polyline. A polyline is a series of XY points that are connected with a straight line. Only one polyline can be defined with each call to ENVI_DEFINE_ROI. When POLYLINE is set, POINT and POLYGON cannot be set.

POLYGON (optional)

Set this optional keyword to indicate that the X and Y arrays define a polygon. A polygon is a closed area that is defined by a series of XY points connected together. In order to properly close the polygon, the first and last point in XPTS and YPTS must be the same. When POLYGON is set, POINT and POLYLINE cannot be set.
XPTS

Use this keyword to specify an array of X points of the specified type. XPTS is in image pixel coordinates. One of the keywords, POINT, POLYLINEx, or POLYGON, must be set to indicate the type of XPTS.

YPTS

Use this keyword to specify an array of Y points of the specified type. YPTS is in image pixel coordinates. One of the keywords, POINT, POLYLINEx, or POLYGON, must be set to indicate the type of YPTS.

Example

This example will create a square polygon ROI that can be displayed onto the bhtmref.img image. First, start ENVI and open the file bhtmref.img. Once the file is open, issue the ENVI_SELECT command and select the file bhtmref.img. Next, use ENVI>Create>ROI and the returned FID to create an ROI that has the file attributes associated with bhtmref.img. Now, define the square polygon points and add this object to the ROI using ENVIDEFINE>ROI. The resulting ROI can be overlaid onto a displayed image from bhtmref.img. This example uses the following files found in the DATA directory of the ENVI installation:

* bhtmref.img
* bhtmref.hdr

; Choose the bhtmref.img file
; ENVI_SELECT, fid=fid
; Get the samples and lines for this file and create an ROI
; ENVI_FILE_QUERY, fid, ns=ns, nl=nl
roi_id = ENVI_CREATE>ROI(ns=ns, nl=nl, $color=4, name='Square')
; Define the square and add the polygon object to the ROI
; xpts = [100, 200, 200, 100, 100]
ypts = [100, 100, 200, 200, 100]
ENVI_DEFINE>ROI, roi_id, /polygon, $xpts=xpts, ypts=ypts
See Also

ENVI_DEFINE_ROI, ENVI_GET_ROI_IDS, ENVI_GET_ROI_DIMS_PTR, ENVI_GET_ROI_DATA,
ENVI_RESTORE_ROIS, ENVI_CREATE_ROI
ENVI_DELETE_ROIS

Use this program to delete ROIs from within ENVI. The program accepts a list of ROIs to delete, or optionally deletes all ROIs.

Calling Sequence

ENVI_DELETE_ROIS [, ROI_ids]

Arguments

ROI_ids (optional)

The ROI IDs to delete. This is the value returned from ENVI_GET_ROI_IDS or ENVI_CREATE_ROI. You may optionally use the keyword ALL to remove all of the ROIs, instead of specifying the ROI IDs.

Keywords

ALL (optional)

Set this keyword to specify that all ROIs are deleted. If the ALL keyword is set, the ROI_ids parameter is not used.

Example

roi_ids = envi_get_roi_ids()
envi_delete_rois, roi_ids
**ENVI_DISP_QUERY**

This procedure gets display information from the display being queried.

**Calling Sequence**

ENVI_DISP_QUERY, \(Dn\)

**Arguments**

\(DN\)

The display number corresponding to an open display. ENVI Display #1 will have \(DN=0\), Display #2 will have \(DN=1\), etc.

**Keywords**

**COLOR**

Use this keyword to specify a named variable that will contain the display color information. The valid values for COLOR are

- 0 - grayscale image
- 2 - classification image
- 8 - 3 band image

**FID**

Use this keyword to specify a named variable that will contain the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long array of size three. For grayscale and classification images, only the first array element is valid. For RGB displays, each FID array element relates to the red (0), green (1), and blue (2) bands. An invalid file ID is specified as -1.

**NL**

Use this keyword to specify a named variable that will contain the number of lines in the file.

**NS**

Use this keyword to specify a named variable that will contain the number of samples in the file.
POS

Use this keyword to specify a named variable that will contain the band positions of the display data. POS is a long array of size three. For grayscale and classification images only the first array element is valid. For RGB displays, each POS array element relates to the red (0), green (1), and blue (2) bands.

REBIN (optional)

Use this optional keyword to specify a named variable that will contain the resize factor currently being used in the Scroll window. If DN is a valid ENVI display number, the resulting variable ("resize" in this case) will be a floating-point number. If the image display doesn't have a Scroll window (i.e., the entire image fits into the full resolution Image window), but the REBIN keyword is used, a (meaningless) REBIN value will still be returned.

W1 (optional)

Use this optional keyword to specify a named variable that will contain the window IDs for the Image, Zoom, and Scroll windows. If DN is a valid ENVI display number, the resulting variable ("win" in this case) will be a three-element long integer array:

- WIN[0]: The IDL window ID of the Image window.
- WIN[1]: The IDL window ID of the Zoom window.
- WIN[2]: The IDL window ID of the Scroll window.

Note

If there is no scroll window in the specified ENVI display, WIN[2] may be equal to 0 or -1, or it may contain the window ID of an invalid display.

x0 (optional)

Use this optional keyword to specify the x position of the upper left-hand pixel of the Image window in file coordinates (i.e., zero-based numbers).

XDS

Use this keyword to specify a named variable that will contain the x display size in pixels for the display windows. XDS is a three-element long array corresponding to the image, zoom, and scroll windows, respectively.
**y0 (optional)**

Use this optional keyword to specify the y position of the upper left-hand pixel of the Image window in file coordinates (i.e., zero-based numbers).

**YDS**

Use this keyword to specify a named variable that will contain the y display size in pixels for the display windows. YDS is a three-element long array corresponding to the image, zoom, and scroll windows respectively.

**ZFACT (optional)**

Use this optional keyword to specify a named variable that will contain the zoom factor currently being used in the Zoom window. If DN is a valid ENVI display number, the resulting variable (“zoom” in this case) will be a two element integer array:

- ZOOM[0]: The zoom factor for the full resolution Image window, which will always be 1.
- ZOOM[1]: The current zoom factor for the Zoom window.

**Example**

```envi
envi_disp_query, dn, xds=xds, yds=yds, fid=fid, 
\$ color=color
```

**See Also**

ENVI_FILE_QUERY
ENVI_DISPLAY_BANDS

Use this routine to display a grayscale or RGB image in a display window. The image will be displayed in the current display. Optionally, a new display window may be created.

Calling Sequence

ENVI_DISPLAY_BANDS, Fid, Pos

Arguments

Fid

A one or three element long integer array of file IDs indicating the file for each of the grayscale or RGB bands to display. When Fid is a three-element array, the bands represent the red, green, and blue channels, respectively. All of the files listed in the Fid array must have the same number of samples and lines.

Pos

A one or three element long integer array of band positions. Elements of Pos are paired with the elements of the Fid array.

Keywords

NEW (optional)

Set this keyword to specify the creation of a new display window. The default is to use the current display window. If non exists, an new window is created.

IMAGE_OFFSET (optional)

Use this keyword to specify the X and Y offset of the image window. IMAGE_OFFSET is a two-element long array specifying the X and Y offset in screen pixels, respectively.

IMAGE_SIZE (optional)

Use this keyword to specify the X and Y size of the image window. IMAGE_SIZE is a two element long array specifying the X and Y image size in screen pixels, respectively.
SCROLL_REBIN (optional)

Use this keyword to specify the initial resize factor for the scroll window. SCROLL_REBIN is a floating-point number greater than one. The size of the scroll window is determined using the size of the input image and SCROLL_REBIN. The size of the scroll window will be (number of samples / SCROLL_REBIN) for X and (number of lines / SCROLL_REBIN) for Y.

SCROLL_OFFSET (optional)

Use this keyword to specify the X and Y offset of the scroll window. SCROLL_OFFSET is a two-element long array specifying the X and Y offset in screen pixels, respectively.

SCROLL_HIDE (optional)

Set this keyword to specify that the scroll window be initially hidden.

ZOOM_FACTOR (optional)

Use this keyword to specify the initial zoom factor for the zoom window.

ZOOM_HIDE (optional)

Set this keyword to specify that the zoom window be initially hidden.

ZOOM_OFFSET (optional)

Use this keyword to specify the X and Y offset of the zoom window. ZOOM_OFFSET is a two-element long array specifying the X and Y offset in screen pixels, respectively.

ZOOM_SIZE (optional)

Use this keyword to specify the X and Y size of the zoom window. ZOOM_SIZE is a two-element long array specifying the X and Y zoom size in screen pixels, respectively.

Example

This example displays the first three bands of a file, specified by FID, as an RGB combination, in a new display window.

```
envi_display_bands, [fid,fid,fid], [0,1,2], /new
```
See Also

ENVI_DISP_QUERY
ENVI_DOIT

The ENVI_DOIT routine is the interface for all of the ENVI processing routines (the _DOIT routines). Each _DOIT is now a string argument to ENVI_DOIT with the specific keywords following the _DOIT argument. For example, the call to CLASS_DOIT is now ENVI_DOIT, ‘CLASS_DOIT’, …. [CLASS_DOIT keywords].

Calling Sequence

ENVI_DOIT, ‘Routine_Name’

Arguments

Routine_Name

Routine_Name is a string variable specifying the processing routine to execute. ENVI commonly refers to the processing routines as the “doit”. Many of the internal ENVI processing routines are available to the programmer in batch mode. The value of Routine_Name should be one of those listed in Table 9-1. Keywords specific to Routine_Name can be found under the description for the specific routine.

Keywords

NO_CATCH (optional)

Set this optional keyword to disable the ENVI catch mechanism. The ENVI catch provides a generalized mechanism for handling exceptions and errors. By disabling the ENVI catch mechanism, the user then has the option to add their own catch mechanism. If no catch mechanism is used and an error occurs, ENVI will let the program stop execution, print an error message, and revert control to the command line. The ENVI session (interactive or batch) must be continued manually from the command line. See the CATCH routine to add a user-defined catch mechanism. This keyword is used only with the routine ENVI_DOIT.

Routine_Name Keywords

Keywords specific to Routine_Name are found under the description for the routine.

Example

The following example will run statistics on bhtmref.img. First, ENVI is initialized in batch mode and the batch log file is set to batch.txt. Next, the file
bhtmref.img is opened and ENVI_STATS_DOIT is used to perform basic statistical calculations. The routine ENVI_STATS_DOIT is the string argument to ENVI_DOIT. The keywords to ENVI_STATS_DOIT are listed in the call to ENVI_DOIT. After the statistics are completed, the mean value is printed.

This example uses the following files found in the data directory of the ENVI installation:

- bhtmref.img
- bhtmref.hdr

```pro
pro example_envi_doit
; ; First restore all the base save files.
; envi, /restore_base_save_files
; ; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; ; Open the input file
; envi_open_file, 'bhtmref.img', r_fid=fid 
if (fid eq -1) then begin
   envi_batch_exit
   return 
endif 
; ; Get the samples, lines and # bands 
; for the input file.
; envi_file_query, fid, ns=ns, nl=nl, nb=nb 
; ; Set the dims and pos to calculate 
; statistics for all data (spatially and 
; spectrally) in the file.
; dims = [-1l, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
; ; Calculate the basic statistics and the 
; histogram for the input data file. Print 
; out the calculated information.
; envi_doit, 'envi_stats_doit', fid=fid, pos=pos, $
dims=dims, comp_flag=1, mean=mean
;
print, 'Mean', mean
;
; Exit ENVI
;
envi_batch_exit
end

See Also

ENVI, ENVI_BATCH_INIT, ENVI_BATCH.EXIT
ENVI_ENTER_DATA

This procedure is used to enter image data in memory into ENVI. The individual images appear as bands in the Available Bands List and are available for display or use by other ENVI functions. ENVI_ENTER_DATA internally calls ENVI_SETUP_HEAD and registers the bands in the available bands list. Items entered using ENVI_ENTER_DATA are considered IN_MEMORY items.

Calling Sequence

ENVI_ENTER_DATA, Data

Arguments

Data

A two or three-dimensional data array to enter of type Byte, Integer, Unsigned Integer, Long Integer, Unsigned Long Integer, Long 64 Integer, Unsigned Long 64 Integer, Floating Point, Double Precision, Complex, or Double Complex. The data must be in BSQ format and have the dimensions Data[samples,lines] or Data[samples,lines,bands].

Note

The Data array is incorporated into the ENVI session directly—rendering the variable undefined after the call to ENVI_ENTER_DATA. If you wish to use the Data array after the call to ENVI_ENTER_DATA, first make a copy and then set the Data argument equal to the copy.

Keywords

BBL (optional)

Use this optional keyword to specify an array of ones and zeros representing the good and bad bands. The number of elements in BBL must be equal to the number of bands in the image. A one represents a good band and a zero represents a bad band.

BNAMES (optional)

Use this optional keyword to specify the band names assigned to the data. BNAMES is a string array of number-of-bands band names. The default band names are [band 1, band 2], etc.
CLASS_NAMES (optional)

Use this keyword to specify a string array of class names for classification images. The first element (class 0) is the “Unclassified” class. CLASS_NAMES must be set when entering classification images.

DEF_STRETCH (optional)

Use this optional keyword to specify the default stretch information. Set DEF_STRETCH equal to the value returned from ENVI_DEFAULT_STRETCH_CREATE.

DESCRIP (optional)

Use this optional keyword to specify a text description of the data, or of the type of processing performed.

FILE_TYPE (optional)

Use this optional keyword to specify an integer value indicating the file type. See “ENVI_FILE_TYPE” on page 417 for details on how to determine the integer value of a file type.

FUNC_COMPLEX (optional)

Set this optional keyword to indicate the Complex Lookup Function to determine which image to display for complex data. This keyword should be set to one of the following:

- 0 - Power (log10 of magnitude)
- 1 - Magnitude (square root of sum of the squares of the real and imaginary)
- 2 - Real (real portion of number)
- 3 - Imaginary (imaginary portion)
- 4 - Phase (arc tangent of imaginary divided by real)

The default image is Power. Only set this keyword if the IDL data type of the image is 6=complex (2 x 32-bit) or 9=double-precision complex (2 x 64-bit)

FWHM (optional)

Use this optional keyword to specify a floating point array of full-width-half-maximum responses for each band. The number of elements in this array is equal to the number of bands in the image.
GEO_POINTS (optional)

Use this optional keyword to specify a 16-element double array of geographic coordinates for the four corners. The array is comprised of four groups of X and Y pixel locations and the corresponding latitude and longitude values (negative for South latitude and negative for West longitude). The four groups of points represent the four corners: upper left, upper right, lower left, lower right. The array is defined as follows:

- `GEO_POINTS[0:3]` -> [Y, LAT, LON] - upper left

INFO (optional)

Use this optional keyword to specify user-defined information. INFO is used to store information which can be passed to user’s spatial and spectral readers. INFO is retrieved from `ENVI_FILE_QUERY`, using the keyword H_INFO, which is a handle to the data. Use HANDLE_VALUE and the handle H_INFO to retrieve the data INFO.

INHERIT (optional)

Use this optional keyword to specify the file inheritance. Set INHERIT equal to the value returned from `ENVI_SET_INHERITANCE`.

LOOKUP (optional)

Use this keyword to specify a long array of class RGB values for classification images. The LOOKUP array contains one RGB triple for each class specified with the NUM_CLASSES keyword. The dimensions of the array are [3, NUM_CLASSES] and the RGB triplet is ordered [R, G, B]. LOOKUP must be set when entering a classification image.

MAP_INFO (optional)

Use this optional keyword to specify map information. Set MAP_INFO equal to the value returned from `ENVI_MAP_INFO_CREATE`.

NUM_CLASSES (optional)

Use this optional keyword to specify the number of classes for classification images. Remember to include the “Unclassified” class (class 0) in the number of classes. NUM_CLASSES only needs to be set when entering a classification image.
PIXEL_SIZE (optional)
Use this optional keyword to specify the pixel size of images that are not georeferenced. PIXEL_SIZE is a two-element floating-point array specifying the X and Y pixel sizes, respectively.

R_FID (optional)
Use this optional keyword to specify a named variable that will contain the file ID of the entered data.

SENSOR_TYPE (optional)
Use this optional keyword to specify an integer value indicating the sensor type. See “ENVI_SENSOR_TYPE” on page 504 for details on how to determine the integer value of a sensor type.

SPEC_NAMES (optional)
Use this optional keyword to specify a string array of spectral library names. SPEC_NAMES only needs to be set when entering spectral library files.

UNITS (optional)
Use this optional keyword to specify the PIXEL_SIZE units for images that are not georeferenced. UNITS is an integer value returned from ENVI_TRANSLATE_PROJECTION_UNITS. Georeferenced images do not use this value. Instead, they use the pixel size and units contained in the map information structure.

WL (optional)
Use this optional keyword to specify an array of wavelength values. The number of elements in this array is equal to the number of bands.

XSTART (optional)
Use this optional keyword to specify the X starting sample for the first pixel in the file. The default is zero. XSTART in conjunction with YSTART are used to preserve the spatial reference for subsetted files. When processing a file, the XSTART of the output file is typically set to the XSTART of the input file plus the value of DIMS[1] (the starting sample).


**YSTART (optional)**

Use this optional keyword to specify the Y starting line for the first pixel in the file. The default is zero. YSTART in conjunction with XSTART are used to preserve the spatial reference for subsetted files. When processing a file, the YSTART of the output file is typically set to the YSTART of the input file plus the value of DIMS[3] (the starting line).

**ZPLOT_AVERAGE (optional)**

Use this optional keyword to specify a two-element long array for the X and Y window size (in pixels) for the Z-Profile. The window size must be “1” or greater. The Z-Profile is formed from the average of the profiles within the specified window. The default window size is [1,1].

**ZRANGE (optional)**

Use this optional keyword to specify a two-dimensional array for the lower and upper range to be used by default in spectral plots.

**ZPLOT_TITLES (optional)**

Use this optional keyword to specify a two-element string array for the X and Y plot titles. The default X title is “Band Number” for images with no wavelength information and “Wavelength” for images with wavelength information. The default Y axis title is “Value”.

**Examples**

The following example will generate 256x256 byte ramp using the function BINDGEN. This array will then be entered into ENVI using the ENVI_ENTER_DATA routine. This is the simplest use of ENVI_ENTER_DATA.

```python
data = BINDGEN(256,256)
ENVI_ENTER_DATA, data
```

The following example will create two classes (plus the Unclassified class) from the ramp image and enter the resulting classification image into ENVI. The Unclassified class will be Black [0,0,0], the first class will be Red [255,0,0] and the second class will be Yellow [255,255,0]. The class names will be Unclassified, Red, and Yellow. The classification image will have the description “Example Classification Image” and the band name “Ramp Classification.”

```python
; Create a 2D ramp and then classify all values
; from 20 to 100 in the first class (classification
```
; data value equal to one) and classify all values
; from 101 to 220 into the second class
; (classification data value equal to two)
;
data = BINDGEN(256,256)
class = BYTE((data ge 20 and data le 100) + $
               2B * (data ge 101 and data le 220))

; Create the classification information
;
class_names = ['Unclassified','Red','Yellow']
lookup = [[0,0,0],[255,0,0],[255,255,0]]
bnames = ['Ramp Classification']
descip = 'Example Classification Image'
file_type = ENVI_FILE_TYPE('ENVI Classification')
;
Enter the data into ENVI
;
ENVI_ENTER_DATA, class, num_classes=3, $
   class_names=class_names, lookup=lookup, $
   file_type=file_type, bnames=bnames, $
   descr=descip

The following example will take the ramp image and assign a geographic projection
to the image with the upper left corner at 15 degrees south, 48 degrees west, and an X
and Y pixel size of one arc second. The map projection will be created using
ENVI_MAP_INFO_CREATE and the resulting structure will use the MAP_INFO
keyword when entering the data into ENVI.


;
;
;
data = BINDGEN(256,256)
;
;
cmc = [.5D,.5, -48,-15]
ps = [1D/3600, 1D/3600]
units = ENVI_TRANSLATE_PROJECTION_UNITS('Degrees')
map_info = ENVI_MAP_INFO_CREATE(/geographic, $
   mc=mc, ps=ps, units=units)
;
; Enter the data into ENVI
;
ENVI_ENTER_DATA, data, map_info=map_info
See Also

ENVI_SETUP_HEAD
**ENVI_EVF_CLOSE**

Use this routine to close an ENVI Vector File (EVF). After an EVF file is opened with ENVI_EVF_OPEN (or the EVF Id is returned from ENVI_EVF_DEFINE_CLOSE) the file must be closed using ENVI_EVF_CLOSE.

**Calling Sequence**

ENVI_EVF_CLOSE, Evf_id

**Arguments**

Evfd_id

The EVF Id returned from ENVI_EVF_OPEN (or the Evf Id returned from ENVI_EVF_DEFINE_INIT).

**Example**

```pro
pro example_envi_evf_info
  ; Open the EVF file
  evf_fname = 'test.evf'
  evf_id = envi_evf_open(evf_fname)
  ; Get the vector information
  envi_evf_info, evf_id, num_recs=num_recs, $
    data_type=data_type, projection=projection, $
    layer_name=layer_name
  ; Print the information about the
  ; vector file.
  print,'File: ', evf_fname
  print,'Layer name: ', layer_name
  print,'Num Recs : ', num_recs
  print,'Data Type : ', data_type
  print,'Projection: ', projection.name
  ; Close the EVF file
  envi_evf_close, evf_id
end
```
See Also

ENVI_WRITE_DBF_FILE, ENVI_EVF_DEFINE_INIT,
ENVI_EVF_DEFINE_ADD_RECORD, ENVI_EVF_DEFINE_CLOSE,
ENVI_EVF_OPEN, ENVI_EVF_INFO, ENVI_EVF_READ_RECORD
ENVI_EVF_DEFINE_ADD_RECORD

Use this routine to add a record to a new ENVI Vector File (EVF). The EVF file must be initialized using ENVI_EVF_DEFINE_INIT. ENVI_EVF_DEFINE_ADD_RECORD accepts points, polylines, and polygon records. After all records are defined close the EVF file with ENVI_EVF_DEFINE_CLOSE. The returned value from ENVI_EVF_DEFINE_CLOSE can be used to access the new vectors.

Calling Sequence

ENVI_EVF_DEFINE_ADD_RECORD, Evfd_id, Points

Arguments

Evfd_id
The EVF Id returned from ENVI_EVF_DEFINE_INIT.

Points
An array of XY points to for a given point, polyline or polygon. A point is a single XY pair, a polyline consists of multiple XY pairs, and a polygon is a polyline with the first and last points the same. Points is an array of (2,npts) where (0,*) are the X points and (1,*) are the Y points.

Example

```plaintext
pro example_envi_evf_define_init
; ; Define 10 lat/lon points
; lat_points = 40. + findgen(10)/100.
lon_points = -105 + findgen(10)/100.
; ; Create a Geographic projection
; with the default datum and units.
; proj = envi_proj_create(/geographic)
; ; Initialize the new EVF file
; evfd_id = envi_evf_define_init('test.evf', $
    projection=proj, data_type=4, $
    layer_name='Lat/Lon Points')
```
if (ptr_valid(evfd_id) eq 0) then return 
;
Enter the individual points as records
;
for i=0L,9 do begin
  envi_evf_define_add_record, evfd_id, $
  [lon_points[i],lat_points[i]]
end
;
Close definition and close the EVF file
;
evf_id = envi_evf_define_close(evfd_id)
evi_evf_close, evf_id
end

See Also

ENVI_WRITE_DBF_FILE, ENVI_EVF_DEFINE_INIT,
ENVI_EVF_DEFINE_CLOSE, ENVI_EVF_OPEN, ENVI_EVF_INFO,
ENVI_EVF_READ_RECORD, ENVI_EVF_CLOSE
ENVI_EVF_DEFINE_CLOSE

Use this function to close a new ENVI Vector File (EVF). The EVF file must be initialized using ENVI_EVF_DEFINE_INIT. After defining the new records using ENVI_EVF_DEFINE_ADD_RECORD close the file with ENVI_EVF_DEFINE_CLOSE before accessing the new vectors.

Calling Sequence

\[ \text{Result} = \text{ENVI\_EVF\_DEFINE\_CLOSE}(\text{Evfd\_id}) \]

Arguments

Evfd_id

The EVF Id returned from ENVI_EVF_DEFINE_INIT.

Keywords

RETURN_ID (optional)

Set this keyword to specify that the Id of the newly defined EVF be returned. This keyword saves the step of having to open the new file using ENVI_EVF_OPEN.

Example

```pro
pro example_envi_evf_define_init

; Define 10 lat/lon points
lat_points = 40. + findgen(10)/100.
lon_points = -105 + findgen(10)/100.

; Create a Geographic projection with the default datum and units.
proj = envi_proj_create(/geographic)

; Initialize the new EVF file
evfd_id = envi_evf_define_init('test.evf', $
  projection=proj, data_type=4, $
  layer_name='Lat/Lon Points')
if (ptr_valid(evfd_id) eq 0) then return
```

; Enter the individual points as records
;
for i=0L,9 do begin
   envi_evf_define_add_record, evfd_id, $
   [lon_points[i],lat_points[i]]
end
;
; Close definition and close the EVF file
;
evf_id = envi_evf_define_close(evfd_id)
envi_evf_close, evf_id
end

See Also

ENVI_WRITE_DBF_FILE, ENVI_EVF_DEFINE_INIT,
ENVI_EVF_DEFINE_ADD_RECORD, ENVI_EVF_OPEN, ENVI_EVF_INFO,
ENVI_EVF_READ_RECORD, ENVI_EVF_CLOSE
ENVI_EVF_DEFINE_INIT

Use this function to initialize a new ENVI Vector File (EVF). The returned result is used when defining and closing the new EVF file. The function allows specification of an output filename, layer name, data type and projection.

Calling Sequence

Result = ENVI_EVF_DEFINE_INIT(Filename)

Arguments

Filename

Use this keyword to specify an output file name for the EVF.

Keywords

DATA_TYPE (optional)

Use this keyword to specify the ENVI data type of the file. DATA_TYPE uses the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

Use this keyword to specify a single string name for the EVF layer. Each EVF file consists on only a single layer. If LAYER_NAME is not set the default is to use ‘New Layer’.

PROJECTION (optional)

Use this optional keyword to specify a map projection. PROJECTION is a projection structure returned from ENVI_GET_PROJECTION or ENVI_PROJ_CREATE.

To allocate a projection information structure use:

    t_proj = {envi_proj_struct}

If PROJECTION is not set the default is to use an Arbitrary projection.
Example

```plaintext
pro example_envi_evf_define_init
;
; Define 10 lat/lon points
;
l_lat_points = 40. + findgen(10)/100.
lon_points = -105 + findgen(10)/100.
;
; Create a Geographic projection
; with the default datum and units.
;
proj = envi_proj_create(/geographic)
;
; Initialize the new EVF file
;
evfd_id = envi_evf_define_init('test.evf', $
   projection=proj, data_type=4, $  
   layer_name='Lat/Lon Points')
if (ptr_valid(evfd_id) eq 0) then return
;
; Enter the individual points as records
;
for i=0L,9 do begin
   envi_evf_define_add_record, evfd_id, $  
   [lon_points[i],lat_points[i]]
end
;
; Close definition and close the EVF file
;
evf_id = envi_evf_define_close(evfd_id)
envi_evf_close, evf_id
end
```
ENVI_EVF_INFO

Use this routine to get general information about an ENVI Vector File (EVF). After the EVF file is opened with ENVI_EVF_OPEN (or the EVF Id is returned from ENVI_EVF_DEFINE_CLOSE) use the appropriate keyword to retrieve the number of records, layer name, data type and/or projection.

Calling Sequence

ENVI_EVF_INFO, Evf_id

Arguments

Evf_id

The EVF Id returned from ENVI_EVF_OPEN (or the Evf Id returned from ENVI_EVF_DEFINE_INIT).

Keywords

DATA_TYPE (optional)

Use this keyword to specify a named variable that will contain the ENVI data type of the vector file. DATA_TYPE uses the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

LAYER_NAME (optional)

Use this keyword to specify a named variable that will contain the string name for the EVF layer. Each EVF file consists on only a single layer.

NUM_RECS (optional)

Use this keyword to specify a named variable that will contain the number of vector records in the EVF file.

PROJECTION (optional)

Use this keyword to specify a named variable that will contain the projection information for the EVF data. PROJECTION is a projection structure as defined by ENVI_PROJ_CREATE.
Example

```plaintext
pro example_envi_evf_info
 ;
 ; Open the EVF file
 ;
 evf_fname = 'test.evf'
ev_id = envi_evf_open(evf_fname)
 ;
 ; Get the vector information
 ;
envi_evf_info, ev_id, num_recs=num_recs, $
 data_type=data_type, projection=projection, $
 layer_name=layer_name
 ;
 ; Print the information about the
 ; vector file.
 ;
 print,'File: ', evf_fname
 print,'Layer name: ', layer_name
 print,'Num Recs : ', num_recs
 print,'Data Type : ', data_type
 print,'Projection: ', projection.name
 ;
 ; Close the EVF file
 ;
envi_evf_close, ev_id
end
```

See Also

ENVI_WRITE_DBF_FILE, ENVI_EVF_DEFINE_INIT,
ENVI_EVF_DEFINE_ADD_RECORD, ENVI_EVF_DEFINE_CLOSE,
ENVI_EVF_OPEN, ENVI_EVF_READ_RECORD, ENVI_EVF_CLOSE
ENVI_EVF_OPEN

Use this function to open an existing ENVI Vector File (EVF). This function is for existing EVF files. To define a new EVF file use ENVI_EVF_DEFINE_INIT. After opening the EVF file ENVI_EVF_INFO will return general information and ENVI_EVF_GET_RECORD will return specific EVF records. When finished accessing the EVF file use ENVI_EVF_CLOSE to close the file.

Calling Sequence

Result = ENVI_EVF_OPEN(Filename)

Arguments

Filename

Use this keyword to specify the EVF file name.

Example

pro example_envi_evf_info
;
; Open the EVF file
;
evf_fname = 'test.evf'
evf_id = envi_evf_open(evf_fname)
;
; Get the vector information
;
envi_evf_info, evf_id, num_recs=num_recs, $
   data_type=data_type, projection=projection, $
   layer_name=layer_name
;
; Print the information about the
; vector file.
;
print,'File: ', evf_fname
print,'Layer name: ', layer_name
print,'Num Recs : ', num_recs
print,'Data Type : ', data_type
print,'Projection: ', projection.name
;
; Close the EVF file
;
envi_evf_close, evf_id
end
See Also

ENVI_WRITE,DBF_FILE, ENVI_EVF_DEFINE_INIT,
ENVI_EVF_DEFINE_ADD_RECORD, ENVI_EVF_DEFINE_CLOSE,
ENVI_EVF_INFO, ENVI_EVF_READ_RECORD, ENVI_EVF_CLOSE
ENVI_EVF_READ_RECORD

Use this function to retrieve records from an ENVI Vector File (EVF). After the EVF file is opened with ENVI_EVF_OPEN (or the EVF Id is returned from ENVI_EVF_DEFINE_CLOSE) the specified record can be retrieved. The routine ENVI_EVF_INFO is used to get the number of records.

Calling Sequence

Result = ENVI_EVF_READ_RECORD(Evf_id, Record_number)

Arguments

Evfd_id

The EVF Id returned from ENVI_EVF_OPEN (or the Evf Id returned from ENVI_EVF_DEFINE_INIT).

Record_number

The record number for the vector to retrieve. Record_number is a long value from zero to NUM_RECS-1. The NUM_RECS value is retrieved using ENVI_EVF_INFO.

Example

pro example_envi_evf_read_record
;
; Open the EVF file
;
evf_fname = 'test.evf'
evf_id = envi_evf_open(evf_fname)
;
; Get the number of records in
; the vector file.
;
envi_evf_info, evf_id, num_recs=num_recs
;
; Print out each record
;
for i=0L, num_recs-1 do begin
  rec = envi_evf_read_record(evf_id, i)
  print, i, rec
endfor
;
; Close the EVF file
;
envi_evf_close, evf_id
end

See Also

ENVI_WRITE_DBF_FILE, ENVI_EVF_DEFINE_INIT,
ENVI_EVF_DEFINE_ADD_RECORD, ENVI_EVF_DEFINE_CLOSE,
ENVI_EVF_OPEN, ENVI_EVF_INFO, ENVI_EVF_CLOSE
**ENVI_FILE_MNG**

This procedure is used to manage ENVI files in memory and on disk.

**Calling Sequence**

ENVI_FILE_MNG

**Keywords**

**DELETE** (optional)

Set this optional keyword to delete the specified file from the hard disk.

**ID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**REMOVE**

Set this keyword to remove the specified file from within ENVI.

**Example**

Remove the file specified by FID from ENVI and delete it from the disk.

```
envi_file_mng, id=fid, /remove, /delete
```

**See Also**

ENVI_GET_FILE_IDS
ENVI_FILE_QUERY

This procedure gets file information from the header file of the file being queried. If no header exists then the standard ENVI widget for entering file header information appears, the user enters the pertinent information, and execution of the function is continued by clicking on the Accept button.

Calling Sequence

ENVI_FILE_QUERY, Fid

Arguments

Fid

The file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. Fid is a long integer with a value greater than zero. An invalid file ID is specified as -1.

Keywords

BBL (optional)

Use this optional keyword to specify a named variable that will contain an array of ones and zeros representing the bad band list. A “1” represents a good band and a “0” represents a bad band. If no bad bands list is available for the data, the value –1 is returned. Otherwise, the number of elements of BBL is equal to the number of bands, NB.

BNAMES (optional)

Use this optional keyword to specify a named variable that will contain the band names associated with each band.

BYTE_SWAP (optional)

Use this optional keyword to specify a named variable that will contain the byte swap value for the data. If BYTE_SWAP is set, then the byte order of the data specified by FID is different than the current processor. User specified spatial and spectral read routines will need to BYTEORDER the data after reading from disk.
CLASS_NAMES (optional)

Use this optional keyword to specify a named variable that will contain the class names for classification images. CLASS_NAMES is only set when the queried file is type ENVI Classification, otherwise the value is undefined.

DATA_TYPE (optional)

Use this optional keyword to specify a named variable that will contain the IDL data type of the file, using the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

DEF_ZRANGE (optional)

Use this optional keyword to specify a named variable that will contain a two-dimensional array equal to the lower and upper plot ranges used as the default in spectral plots.

DEF_STRETCH (optional)

Use this optional keyword to specify a named variable that will contain the default stretch. The default stretch is used when displaying a band from the file. DEF_STRETCH is the structure created by ENVI_DEFAULT_STRETCH_CREATE.

DESCRIP (optional)

Use this optional keyword to specify a named variable that will contain a string description of the file.

FILE_TYPE (optional)

Use this optional keyword to specify a named variable that will contain the integer file type value. See “ENVI_FILE_TYPE” on page 417 for details on how to test for a given file type.

FNAME (optional)

Use this optional keyword to specify a named variable that will contain the file name, including the path, associated with the file. For memory items, the FNAME contains a string indicating the memory item number and its associated spatial and spectral dimensions.
Chapter 9: ENVI Routines

ENVI Programmer's Guide

ENVI_FILE_QUERY

**FUNC_COMPLEX (optional)**

Set this optional keyword to specify a named variable that will indicate the COMplex Lookup FUCTION. The returned value will be one of the following:

- 0 - Power (log10 of magnitude)
- 1 - Magnitude (square root of sum of the squares of the real and imaginary)
- 2 - Real (real portion of number)
- 3 - Imaginary (imaginary portion)
- 4 - Phase (arc tangent of imaginary divided by real)

Set this keyword only if the IDL data type of the image is 6=complex (2 x 32-bit) or 9=double-precision complex (2 x 64-bit)

**FWHM (optional)**

Use this optional keyword to specify a named variable that will contain an array of full-width-half-maximum values, one for each band. If there is no FWHM associated with the file then -1 is returned.

**H_INFO (optional)**

Use this optional keyword to specify a named variable that will contain a handle pointer to user-defined header information. Use the routine HANDLE_VALUE to retrieve the information. The value of the handle is set equal to the INFO keyword in ENVI_SETUP_HEAD or ENVI_ENTER_DATA.

**INTERLEAVE (optional)**

Use this optional keyword to specify a named variable that will contain the file’s interleave type. If the file is in BSQ format, this variable is set to zero. If BIL format, it is set to one. If BIP format, it is set to two.

**LOOKUP (optional)**

Use this optional keyword to specify a named variable that will contain the RGB lookup values for each class in a classification image. The returned result is a byte array [3,NUM_CLASSES]. LOOKUP is only set when the queried file is type ENVI Classification, otherwise the value -1 is returned.

**LUT_NAME (optional)**

Use this optional keyword to specify a named variable that will contain the filename of the lookup table for the data.
**NB (optional)**

Use this optional keyword to specify a named variable that will contain the number of bands in the file.

**NL (optional)**

Use this optional keyword to specify a named variable that will contain the number of lines in the file.

**NS (optional)**

Use this optional keyword to specify a named variable that will contain the number of samples in the file.

**NUM_CLASSES (optional)**

Use this optional keyword to specify a named variable that will contain the number of classes for classification images. NUM_CLASSES is only set when the queried file is type ENVI Classification, otherwise the value 0 is returned.

**OFFSET (optional)**

Use this optional keyword to specify a named variable that will contain the offset in bytes to the start of the data in the file.

**READ_PROCEDURE (optional)**

Use this optional keyword to specify a named variable that will contain a two-element string array of the procedure names for the spatial and spectral readers, respectively. The ENVI read procedures provide for a powerful mechanism for importing custom formats or files directly into ENVI without the need for conversion. All spatial or spectral requests for data go through the specified read routines.

**SENSOR_TYPE (optional)**

Use this optional keyword to specify a named variable that will contain the integer sensor type value. See “ENVI_SENSOR_TYPE” on page 504 for details on how to test for a given sensor type.

**SNAME (optional)**

Use this optional keyword to specify a named variable that will contain the shortened file name (without the path). For memory items, SNAME contains an empty string.
**SPEC_NAMES (optional)**

Use this optional keyword to specify a named variable that will contain a string array of spectral library names. SPEC_NAMES is only set when the queried file is a spectral library.

**STA_NAME (optional)**

Use this optional keyword to specify a named variable that will contain the statistics file name. If no filename is specified the STA_NAME is equal to the empty string, ‘’.

**WL (optional)**

Use this optional keyword to specify a named variable that will contain an array of wavelength values, one for each band. If there is no wavelength associated with the file, then -1 is returned.

**XSTART (optional)**

Use this optional keyword to specify a named variable that will contain the X starting sample for the first pixel in the file.

**YSTART (optional)**

Use this optional keyword to specify a named variable that will contain the Y starting row for the first pixel in the file.

**Example**

This example prompts the user for a file using ENVI_SELECT. Next, ENVI_FILE_QUERY is used to retrieve the file information for the number of samples, lines and bands, the filename, data type, and interleave. The results are then printed to the screen.

```
ENVI_SELECT, fid=fid

ENVI_FILE_QUERY, fid, ns=ns, nl=nl, nb=nb, $
  fname=fname, data_type=data_type, $
  interleave=interleave

print, 'Filename = ', fname
print, 'ns= ', ns, ' nl= ', nl, ' nb = ', nb
print, 'interleave = ', interleave
print, 'data type = ', data_type
```
See Also

ENVI_DISP_QUERY, ENVI_ENTER_DATA, ENVI_SETUP_HEAD
ENVI_FILE_TYPE

ENVI files may conform to any of a range of different file types. This function returns a file type descriptor. All ENVI files that require special display or input processing have a unique file type (Meta File, Virtual Mosaic, Classification, Spectral Library, FFT Result). Any external file that requires an input read routine also has a designed file type.

Calling Sequence

\[ \text{Result} = \text{ENVI\_FILE\_TYPE} (\text{File\_type}) \]

Arguments

\textbf{File\_type}

\textit{File\_type} can be either an integer or a string. If \textit{File\_type} is an integer, the function returns the string-format description of the file type specified. If \textit{File\_type} is a string, the function returns the integer code describing the file type specified.

Table 9-3 shows the available file types. String file type descriptions are case-sensitive and may contain spaces.

Note

Because additional file types are likely to be added in future releases of ENVI, it is strongly recommended that you use string descriptors rather than integer descriptors when referring to file types. See the example below for details on how to do this.

<table>
<thead>
<tr>
<th>File Type</th>
<th>String Descriptor</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVI Standard</td>
<td>ENVI Standard</td>
<td>0</td>
</tr>
<tr>
<td>ENVI Meta File</td>
<td>ENVI Meta File</td>
<td>1</td>
</tr>
<tr>
<td>ENVI Virtual Mosaic</td>
<td>Virtual Mosaic</td>
<td>2</td>
</tr>
<tr>
<td>ENVI Classification</td>
<td>Classification</td>
<td>3</td>
</tr>
<tr>
<td>ENVI Spectral Library</td>
<td>Spectral Library</td>
<td>4</td>
</tr>
<tr>
<td>ENVI Fast Fourier Transform</td>
<td>FFT Result</td>
<td>5</td>
</tr>
</tbody>
</table>

\textit{Table 9-3: ENVI file types and their string and integer type descriptors.}
Keywords

None

Example

The following example prints the file type string of the file associated with the file ID $fid$. The first command stores the integer file type descriptor in the variable $file\_type$:

```
envi_file_query, fid, file_type=file\_type
```

The next line stores the string file type descriptor in the variable $TYPE\_STRING$:

```
type\_string = envi\_file\_type(file\_type)
```
print, type_string

The following example enters the array DATA into ENVI and sets the file type to SPOT data. The first line gets the integer file type for SPOT data. The second line enters the array.

```
file_type = envi_file_type('Spot CD')
envi_enter_data, DATA, file_type=file_type
```

See Also

ENVI_FILE_QUERY, ENVI_SENSOR_TYPE
**ENVI_FILTER_DOIT**

Use this program to build an FFT filter image.

### Calling Sequence

```
ENVI_DOIT, ‘ENVI_FILTER_DOIT’
```

### Keywords

#### ANN_NAME

Use this keyword to specify an annotation file name for user defined filters. This keyword is not used for circular or band pass filters.

#### FILTER

Use this keyword to specify an integer value corresponding to the filter type to build. Choose one of the following:

- 0 - Circular Pass Filter
- 1 - Circular Cut Filter
- 2 - Band Pass Filter
- 3 - Band Cut Filter
- 4 - User Defined Pass Filter
- 5 - User Defined Cut Filter

#### IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

#### NBP

Use this keyword to specify the number of border pixels for the filter transitions.

#### NS

Use this keyword to specify the number of samples in the output filter image.

#### NL

Use this keyword to specify the number of lines in the output filter image.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

RADIUS

For circular filters, this keyword specifies the filter radius in pixels. For band pass filters, this keyword is a two-element array specifying the inner and outer filter radius in pixels. This keyword is not used for user defined filters.

Example

pro example_envi_filter_doit
    
    ; First restore all the base save files.
    ;
    ; envi, /restore_base_save_files
    ;
    ; Initialize ENVI and send all errors and warnings to the file batch.txt
    ;
    ; envi_batch_init, log_file='batch.txt'
    ;
    ; Set the necessary variables
    ;
    ; ns = 512L
    ; nl = 512L
    ; radius = [60,150]
    ; out_name = 'testfilter.img'
    ;
    ; Call the doit
    ;
    ; envi_doit, 'envi_filter_doit', $ 
    ;     ns=ns, nl=nl, filter=2, $ 
    ;     radius=radius, nbp=0, $ 
    ;     out_name=out_name, r_fid=r_fid 
    ;
    ; Exit ENVI
    ;
    envi_batch_exit
end
ENVI_GET_CONFIGURATION_VALUES

Use this function to get the current setting for all the ENVI configuration items. This function returns an anonymous structure of the current configuration settings. The tag names of the structure are the configuration item and the values are the current setting.

**Calling Sequence**

\[ Result = \text{ENVI\_GET\_CONFIGURATION\_VALUES} \]

**Example**

This example retrieves the current ENVI configuration values.

```plaintext
    cfg = envi_get_configuration_values()
    help, cfg, /structure
```
### ENVI_GET_DATA

Use this function to retrieve spatial image data for any open file. The DIMS keyword allows full control over the spatial dimensions of the returned data allowing retrieval of both the full band or any spatial subset. The band is specified with the POS keyword and only a single band is returned. Additional optional parameters allow automatic resampling of the spatial data to smaller or larger pixels sizes. This function will work with any image open in ENVI regardless of the format or physical storage order of the file.

This routine, along with ENVI_GET_SLICE, can be used in place of tiled processing. However, requests for large images may be limited by the amount of your system RAM available.

#### Calling Sequence

```plaintext
Result = ENVI_GET_DATA()
```

#### Keywords

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-elements array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned by the R_FID keyword to the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.
INTERP (optional)

Use this optional keyword to specify an integer value corresponding to the interpolation type. The keyword is only used when either XFACTOR or YFACTOR is not equal to 1. Choose one of the following.

- 0 - Nearest Neighbor
- 1 - Bilinear
- 2 - Cubic Convolution
- 3 - Pixel Aggregate

POS

Use this keyword to specify the band position of the returned data. POS is a long integer with a value from zero to number-of-bands minus one.

XFACTOR (optional)

Use this optional keyword to specify the X magnification factor for the data. A value of one does not change the data. Values greater than one cause the size to increase, values less than one cause the size to decrease.

YFACTOR (optional)

Use this optional keyword to specify the Y magnification factor for the data. A value of one does not change the data. Values greater than one cause the size to increase, values less than one cause the size to decrease.

Example

This example will open a file, determine its spatial size, and request that all of the data for the first band be stored in the variable DATA. The routine ENVI_OPEN_FILE is used to programmatically open the desired file. Subsequent references to this file will use the file ID returned in the keyword R_FID. Once the file is open ENVI_FILEQUERY is used to determine the number of samples and number of lines. The samples and lines are used to set the DIMS keyword for the function ENVI_GET_DATA. A call to ENVI_GET_DATA with the full band specified by DIMS and the first band specified by POS returns all image data for the first band in a Samples-by-Lines 2-D array.

```
ENVI_OPEN_FILE, 'can_tmr.img', r_fid=fid
ENVI_FILE_QUERY, fid, ns=ns, nl=nl
dims = [-1, 0, ns-1, 0, nl-1]
data = ENVI_GET_DATA(fid=fid, dims=dims, pos=0)
```
See Also

ENVI_GET_IMAGE, ENVI_INIT_TILE, ENVI_OPEN_FILE, ENVI_GET_SLICE
ENVI_GET_DISPLAY_NUMBERS

Use this function to get a list of display numbers. The optional keywords can be used to specify displays that are loaded, only displays with color images, and/or only displays with georeferenced images.

Calling Sequence

```
Result = ENVI_GET_DISPLAY_NUMBERS()
```

Keywords

COLOR (optional)

Set this keyword to specify that the function return only displays with RGB color images. This keyword can be used in conjunction with the other keywords.

GEOREF (optional)

Set this keyword to specify that the function return only displays with georeferenced images. This keyword can be used in conjunction with the other keywords.

LOADED (optional)

Set this keyword to specify that the function return only displays which are loaded with an image. This keyword can be used in conjunction with the other keywords.
ENVI_GET_FILE_IDS

This function returns an array of file IDs for all open ENVI files. A file ID is a long integer with a value greater than zero. If no files are open then a -1 is returned.

Calling Sequence

Result = ENVI_GET_FILE_IDS()

Keywords

None

Example

This example get all the file IDs and prints out the corresponding filename.

```
fids = envi_get_file_ids()
if (fids(0) eq -1) then return
for i=0, n_elements(fids)-1 do begin
    envi_file_query, fids(i), fname=fname
    print, fname
endfor
```

See Also

ENVI_FILE_MNG, ENVI_FILE_QUERY, ENVI_OPEN_FILE
ENVI_GET_IMAGE

This function returns the display data for the associated display. ENVI_GET_IMAGE returns either one band, \((ns, nl)\), or three bands, \((ns, nl, 3)\).

**Calling Sequence**

\[ \text{Result} = \text{ENVI_GET_IMAGE()} \]

**Keywords**

**BAND_POS**

Use this keyword to specify the band position of an RGB image to return. If left undefined then all three RGB bands are returned. For grayscale images set BAND_POS equal to zero.

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-elements array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**DN**

Use this keyword to specify the display number for the data request. Display numbers can be retrieved in the event handler of user managed display events. Retrieve the uvalue for ev.top:

\[ \text{widget}._\text{control, ev.top, get}_\text{uvalue}=dn \]

For user managed display routines just add a menu item to the `display.men` file, see Appendix B of the Users Guide and Appendix A for more details.
**Example**

Retrieve the all the data in the image window. Check to see what type of image is display and set BAND_POS accordingly.

```plaintext
widget_control, ev.top, get_uvalue=dn
envi_disp_query, dn, xds=xds, yds=yds, color=color
if (color eq 0) then band_pos = 0
data=envi_get_image(dn=dn, band_pos=band_pos, $
dims=[-1, 0, xds(0)-1, 0, yds(0)-1])
```

**See Also**

ENVI_DISP_QUERY
ENVI_GET_MAP_INFO

Use this function to get the map information for the specified file or display. The function returns a map information structure. If there is not map information associated with the FID or DN then the returned value is the map information for an Arbitrary projection with an X and Y pixel size of one.

Calling Sequence

Result = ENVI_GET_MAP_INFO()

Keywords

DN (optional)
Use this keyword to specify the display number for the returned projection information. The function will return the map information structure for the data in the display. If the displayed data is not georeferenced an Arbitrary projection is returned.

FID (optional)
Use this keyword to specify the file ID for the returned map information. The function will return the map information structure for the specified file. If the file is not georeferenced an Arbitrary projection is returned. FID is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

Example

; Select an input file
envi_select, fid=fid

; Get the associated map information structure
map_info = envi_get_map_info(fid=fid)

See Also

ENVI_MAP_INFO_CREATE, ENVI_FILE_QUERY, ENVI_DISP_QUERY
ENVI_GET_PROJECTION

Use this function to get the projection information for the specified file, display or map information handle. The function returns the projection information in the ENVI_PROJ_STRUCT structure.

Calling Sequence

Result = ENVI_GET_PROJECTION()

Keywords

DN (optional)

Use this keyword to specify the display number for the returned projection information. The function will return the ENVI_PROJ_STRUCT projection structure for the data in the display. If the displayed data is not georeferenced an Arbitrary projection is returned.

FID (optional)

Use this keyword to specify the file ID for the returned projection information. The function will return the ENVI_PROJ_STRUCT projection structure for the specified file. If the file is not georeferenced an Arbitrary projection is returned. FID is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

PIXEL_SIZE (optional)

Use this keyword to specify a named variable that will contain the X and Y pixel size of the returned projection. The returned pixel size is a two element double precision array. The first element is the X pixel size and the second element is the Y pixel size.

UNITS (optional)

Use this keyword to specify a named variable that will contain the projection units. This is the integer units value and can be transformed to the string units name using the procedure ENVI_TRANSLATE_PROJECTION_UNITS.

Example

```
pro atest
  ; Select a file   envi_select, fid=fid   if (fid eq -1) then
  return
```
; Get the projection information
proj = envi_get_projection(fid=fid)
print, proj
end
ENVI_GET_RGB_TRIPLETS

Use this procedure to get the RGB triplets associated with a graphics color index.

Calling Sequence

ENVI_GET_RGB_TRIPLETS, Index, R, G, B

Arguments

Index

Index is the graphics color index for the desired triplet. The modulo operator using the total number of graphics colors is applied to Index prior accessing the color triplets.

R

R is the red color value.

G

G is the red color value.

B

B is the red color value.

Example

pro example_envi_get_rgb_triplets
    ;
    ; Save the RGB values for 5 graphics colors into the array lookup
    ;
    lookup = bytarr(3,5)
    ;
    for i=0L,4 do begin
        envi_get_rgb_triplets, i+2, r, g, b
        lookup[0,i] = [r,g,b]
    endfor
    ;
    ; Print the resulting array
    ;

print, lookup
end
ENVI_GET_ROI

This function is used to get the address of each point in a Region of Interest (ROI). ENVI_GET_ROI returns the ROI pointers associated with the input ID. ROI addresses are a one dimensional spatial address. The address of a ROI pixel at \((r_{sample}, r_{line})\) would be calculated as, \(r_{sample} + r_{line} \times ns\).

Calling Sequence

\[
\text{Result} = \text{ENVI_GET_ROI}(\text{ROI\_Id})
\]

Arguments

ROI\_Id

\(\text{ROI\_Id}\) is a single ID returned from the function ENVI_GET_ROI_IDS.

Keywords

ROI\_COLOR

Use this keyword to specify a named variable that will contain the RGB color value for the \(\text{ROI\_Id}\). ROI\_COLORS is byte array of size three.

ROI\_NAME

Use this keyword to specify a named variable that will contain the ROI name. ROI\_NAME is a string.

Example

Use ENVI_GET_ROI to get the address and name of each region of interest associated with display DN. Then print out the ROI name and number of points.

\[
\begin{align*}
\text{roi\_ids} &= \text{envi\_get\_roi\_ids}(\text{dn}=\text{dn}) \\
\text{for} \ i = 0, \ \text{n\_elements}(\text{roi\_ids}) - 1 \ \text{do begin} \\
& \quad \text{roi\_addr} = \text{envi\_get\_roi}(\text{roi\_ids}(i), \ \text{roi\_name}=\text{name}) \\
& \quad \text{print}, \ \text{\'ROI: \', name, n\_elements}(\text{roi\_addr}) \\
\text{endfor}
\end{align*}
\]

See Also

ENVI_GET_ROI\_IDS, ENVI\_GET\_ROI\_DIMS\_PTR, ENVI\_GET\_ROI\_DATA, ENVI\_RESTORE\_ROIS, ENVI\_CREATE\_ROI, ENVI\_DEFINE\_ROI
ENVI_GET_ROI_DATA

This function is used to get the data associated with a roi address.

Calling Sequence

\[ Result = \text{ENVI\_GET\_ROI\_DATA}(ROI\_Id) \]

Arguments

ROI_Id

ROI_Id is a single ID returned from the function ENVI_GET_ROI_IDS.

Keywords

ADDR (optional)

Use this keyword to specify the addresses of each ROI data value returned. The ROI addresses are returned from the routine ENVI_GET_ROI. The returned data values match the corresponding input address.

FID

Use this keyword to specify the file ID for the open file. This is the value returned by the R_FID keyword to the ENVI\_OPEN\_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

POS

Use this keyword to specify the band position(s) to get ROI data for. If POS is a single element the result is \[ Result(npts) \] otherwise the result is \[ Result(n\_elements(pos), npts) \].

Example

Use ENVI_GET_ROI_DATA to get the data values for bands zero, one and two.

\[ \text{data} = \text{envi\_get\_roi\_data}(\text{roi\_id}(0), \text{fid=fid, pos}=[0,1,2]) \]
Notes

If you have a large ROI and many bands in the POS array then the memory requirements may exceed the capacity of your machine. One way to break up the problem is to request fewer band in the POS array, processes the data, and then get the ROI for the next set of bands. To minimize memory usage only request one band at a time and then loop on the number of bands.

See Also

ENVI_GET_ROI_IDS, ENVI_GET_ROI_DIMS_PTR, ENVI_RESTORE_ROIS, ENVI_RESTORE_ROIS, ENVI_CREATE_ROI, ENVI_DEFINE_ROI
ENVI_GET_ROI_DIMS_PTR

This function is used to get the DIMS Region of Interest (ROI) pointer value. ENVI_GET_ROI_DIMS_PTR returns the value used in DIMS(0). Since ROI pointer values are likely to change as ROIs are created, update or removed. The user should always use the ROI_ID to get the current ROI pointer value.

Calling Sequence

```
Result = ENVI_GET_ROI_DIMS_PTR(ROI_Id)
```

Arguments

ROI_Id

ROI_Id is a single ID returned from the function ENVI_GET_ROI_IDS.

Example

Set the first element of the DIMS array using ROI_ID(0). To get the Nth DIMS pointer then just use ROI_ID(N).

```
roi_ids = envi_get_roi_ids(fid=fid)
dims = [envi_get_roi_dims_ptr(roi_ids(0), 0, 0, 0, 0]
```

See Also

ENVI_GET_ROI_IDS, ENVI_GET_ROI, ENVI_GET_ROI_DATA, ENVI_RESTORE_ROIS, ENVI_CREATE_ROI, ENVI_DEFINE_ROI
ENVI_GET_ROI_IDS

This function returns the Region of Interest (ROI) IDs associated with a display, a file or a spatial size (ns,nl). ENVI_GET_ROI_IDS returns an array of ROI IDs.

Calling Sequence

Result = ENVI_GET_ROI_IDS()

Keywords

DN (optional)

Use this optional keyword to specify the display number associated with the desired ROIs. Display numbers can be retrieved in the event handler of user managed display events. Retrieve the uvalue for ev.top,

    widget_control, ev.top, get_uvalue=dn.

For user managed display routines just add a menu item to the display.men file, see Appendix A for more details.

FID (optional)

Use this optional keyword to specify the file ID associated with the desired ROIs. This is the value returned by the R_FID keyword to the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

NL (optional)

Use this optional keyword to specify the number of image lines associated with the desired ROIs. If NL is specified then NS must be specified.

NS (optional)

Use this optional keyword to specify the number of image samples associated with the desired ROIs. If NS is specified then NL must be specified.

ROI_COLORS (optional)

Use this optional keyword to specify a named variable that will contain the RGB color value for each ROI ID. ROI_COLORS is byte array of size (3, # ROI IDs).
**ROI_NAMES (optional)**

Use this optional keyword to specify a named variable that will contain a string array of ROI names for each ROI ID. The default name includes the ROI name, ROI color, and number of points. Set the keyword SHORT_NAME or LONG_NAME to modify the default ROI_NAMES.

**SHORT_NAME (optional)**

Set this optional keyword to return only the ROI name in the variable specified by ROI_NAMES. This keyword cannot be set if LONG_NAME is set.

**LONG_NAME (optional)**

Set this keyword to return the ROI name, ROI color, number of point and associated image size in the variable specified by ROI_NAMES. This keyword cannot be set if SHORT_NAME is set.

**Example**

Get the ROI IDs associated with a display. When a routine is called from the display menu the uvalue of `ev.top` contains the display number.

```
    widget_control, ev.top, get_uvalue=dn
    roi_ids = envi_get_roi_ids(dn=dn, roi_names=roi_names, $
        roi_colors=roi_colors)
```

**See Also**

`ENVI_GET_ROI, ENVI_GET_ROI_DIMS_PTR, ENVI_GET_ROI_DATA, ENVI_RESTORE_ROIS, ENVI_CREATE_ROI, ENVI_DEFINE_ROI`
ENVI_GET_SLICE

This function returns the requested spectral slice of data. The spectral slices can be returned in either BIP or BIL storage order.

Calling Sequence

Result = ENVI_GET_SLICE()

Keywords

BIL
Set this keyword to specify that the data should be returned in BIL format.

BIP
Set this keyword to specify that the data should be returned in BIP format.

FID
Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

POS
Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

LINE
Use this keyword to specify the line number to extract the slice from. LINE is a zero based line number.

XE
Use this keyword to specify the X ending value. XE is a zero based number.

XS
Use this keyword to specify the X starting value. XS is a zero based number.
Example

Return line 20 (zero based count), pixels 20 through 40, bands 1 and 3.

```python
data = envi_get_slice(fid=fid, line=20, pos=[1,3], xs=20, xe=40)
```
ENVI_GET_STATISTICS

This procedure is used to read data from an ENVI statistics (.sta) file. The values from the statistics file are returned by the keyword arguments. The value of POS and CPOS, depending on the statistics retrieved, indicate which bands have valid statistics values.

Calling Sequence

ENVI_GET_STATISTICS, STA_Name

Arguments

STA_name

STA_Name is the filename of an ENVI statistics (.sta) file.

Keywords

COV

Set this keyword to specify a named variable that will contain the covariance for the image. If COV is undefined the covariance was not calculated. COV is a (nb,nb) array where nb is defined by CPOS

CPOS

Set this keyword to specify a named variable that will contain the array of band indexes used to calculate COV, EVAL and EVEC. The value of CPOS can even be [-1] in which case COV, EVAL and EVEC are not valid.

DMAX

Set this keyword to specify a named variable that will contain the image maximums. If DMAX is undefined the basic statistics were not calculated. DMAX is a (nb) array where nb is defined by POS

DMIN

Set this keyword to specify a named variable that will contain the image minimum. If DMIN is undefined the basic statistics were not calculated. DMIN is a (nb) array where nb is defined by POS
**EVAL**

Set this keyword to specify a named variable that will contain the eigen values for the image. If EVAL is undefined the eigen values was not calculated. EVAL is a \((nb,nb)\) array where \(nb\) is defined by CPOS

**EVEC**

Set this keyword to specify a named variable that will contain the eigen vectors for the image. If EVEC is undefined the eigen vectors were not calculated. EVEC is a \((nb,nb)\) array where \(nb\) is defined by CPOS

**MEAN**

Set this keyword to specify a named variable that will contain the image mean. If MEAN is undefined the basic statistics were not calculated. MEAN is a \((nb)\) array where \(nb\) is defined by POS

**POS**

Set this keyword to specify a named variable that will contain the array of band indexes used to calculate basic statistics, DMIN, DMAX, MEAN and STD. The value of POS can even be \([-1]\) in which case DMIN, DMAX, MEAN and STD are not valid.

**STDV**

Set this keyword to specify a named variable that will contain the image standard deviation. If STDV is undefined the basic statistics were not calculated. STDV is a \((nb)\) array where \(nb\) is defined by POS

**Example**

This example get the statistics from the file `myimage.sta` and prints out the minimum, maximum and mean for each band statistics was computed on.

```plaintext
envi_get_statistics, '/data/myimage.sta', mean=mean, $  
  dmax=dmax, dmin=dmin, pos=pos  
if (pos(0) eq -1) then return  
for i=0, n_elements(pos)-1 do $  
  print, 'Band ', pos(i), dmin(i), dmax(i), mean(i)
```
Notes

The existence of a statistics file does not guarantee that statistics are computed for all bands, always check POS and CPOS. In fact a NULL statistics file may exists where POS=-1 and CPOS=-1.
ENVI_GET_TILE

This function returns the requested tile data. Tiles may be requested multiple times and in any order between the ENVI_INIT_TILE and the ENVI_TILE_DONE calls.

Calling Sequence

\[ Result = \text{ENVI_GET_TILE}(\text{Tile}_\text{id}, \text{Cur}_\text{tile}) \]

Arguments

- **Tile_id**
  The tile id returned from ENVI_INIT_TILE

- **Cur_tile**
  The tile number for the requested tile. This number must fall between zero and \( \text{num}_\text{tiles}-1 \).

Keywords

- **BAND_INDEX**
  If the file in use is in BSQ tile interleave format, use this keyword to specify a named variable that will contain the index to the current band.

- **YE**
  Use this keyword to specify a named variable that will contain the ending Y value for the current tile, in file coordinates.

- **YS**
  Use this keyword to specify a named variable that will contain the starting Y value for the current tile, in file coordinates. For spatially subsetted images, the first YS is equal to the first line in the subset image, which is not necessarily line zero.
Example

This example illustrates the tile processing loop.

```
for i=0, num_tiles-1 do begin
  tile_data=envi_get_tile(tile_id, i, ys=ys)
  .
  tile processing commands here
endfor
```

Notes

When trying to determine the output line number make sure to subtract the spatial subset index dims(3) from the returned YS:

```
ys_memory = ys-dims(3)
```

See Also

ENVI_INIT_TILE, ENVI_TILE_DONE
ENVI_GRID_DOIT

Use this program to convert a set of irregularly gridded XYZ points to a raster image.

Calling Sequence

ENVI_DOIT, ‘ENVI_GRID_DOIT’

Keywords

EXTRAP (optional)

Set this keyword to allow extrapolation to the outer edge of the image. Otherwise the image is defined by the bounding polygon around the exterior points and all other points are set to zero.

I_PROJ (optional)

Use this keyword to specify the input map projection structure for the irregularly gridded arrays of points, X_PTS and Y_PTS. I_PROJ is a structure of type \{envisprojstruct\}, ENVI projection structures are defined in greater detail in Appendix B of the Users Manual.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

INTERP

Set this keyword to specify the type of interpolation. INTERP is set to one of the following

- 0 - Linear interpolation
- 1 - Quintic interpolation

O_PROJ (optional)

Use this keyword to specify the output map projection structure for the gridded points. The points are automatically converted from the input projection to the output projection. O_PROJ is a structure of type \{envisprojstruct\}, ENVI projection structures are defined in greater detail in Appendix B of the Users Manual.
OUT_DT

Use this keyword to specify the output file data type. OUT_DT should be an integer value matching one of the standard data types: 1-byte, 2-integer, 3-long, 4-float, 5-double 6-complex.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

PIXEL_SIZE

Use this keyword to specify the pixel size of output image. PIXEL_SIZE is a floating point array of two specifying the X and Y pixel size respectively.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

XMAX (optional)

Use this keyword to specify the maximum X value to use. The default is to use the maximum of the X_PTS array.

XMIN (optional)

Use this keyword to specify the minimum X value to use. The default is to use the minimum of the X_PTS array.

X_PTS

Use this keyword to specify a corresponding array of irregularly gridded X points. The number of elements of X_PTS, Y_PTS, and Z_PTS must be equal.

YMAX (optional)

Use this keyword to specify the maximum Y value to use. The default is to use the maximum of the Y_PTS array.

YMIN (optional)

Use this keyword to specify the minimum Y value to use. The default is to use the minimum of the Y_PTS array.
Y_PTS

Use this keyword to specify a corresponding array of irregularly gridded Y points. The number of elements of X_PTS, Y_PTS, and Z_PTS must be equal.

Z_PTS

Use this keyword to specify a corresponding array of irregularly gridded Z points. The number of elements of X_PTS, Y_PTS, and Z_PTS must be equal.

Example

```plaintext
forward_function envi_proj_create

pro example_envi_grid_doit
 ;
 ; First restore all the base save files.
 ;
 envi, /restore_base_save_files
 ;
 ; Initialize ENVI and send all errors
 ; and warnings to the file batch.txt
 ;
 envi_batch_init, log_file='batch.txt'
 ;
 ; Set the necessary variables
 ;
 x_pts = [0, 500, 500, 0, 250]
 y_pts = [0, 0, 500, 500, 250]
 z_pts = [0, 100, 200, 300, 1000]
 o_proj = envi_proj_create(/arbitrary)
 pixel_size = [1.,1.]
 out_name = 'testimg'
 ;
 ; Call the doit
 ;
 envi_doit, 'envi_grid_doit', $
 x_pts=x_pts, y_pts=y_pts, $
 z_pts=z_pts, out_dt=2, $
 pixel_size=pixel_size, $
 o_proj=o_proj, extrap=1, $
 out_name=out_name, interp=1, $
 r_fid=r_fid
 ;
 ; Exit ENVI
 ;
 envi_batch_exit
 end
```
ENVI_INFO_WID

Use this procedure to display an ENVI report widget. The contents of the string argument are displayed in a text widget.

Calling Sequence

ENVI_INFO_WID, Str

Arguments

Str

An array of strings to display in the text widget. Each element in the string array is displayed on a separate line.

Keywords

TITLE (optional)

Set this optional keyword to specify the title of the report widget.

YS (optional)

Use this optional keyword to specify the size in lines of the text widget.

Example

This example displays three lines of text with one blank line in a report widget.

str = ['Line 1', 'Next line is blank', '', 'Line 4']
envi_info_wid, str, title='Report'
**ENVI_INIT_TILE**

This function is used to initialize processing of tile data. The returned value is the tile ID used to access the data.

**Calling Sequence**

\[ \text{Result} = \text{ENVI_INIT_TILE}(\text{Fid}, \text{Pos}) \]

**Arguments**

- **Fid**
  
  The file id of the file to process.

- **Pos**
  
  The array of band positions to process.

**Keywords**

- **INTERLEAVE**
  
  Use this keyword to specify the type of tile interleaving to perform. The default is to perform the same type of interleave as the file. Set this keyword equal to zero if the tile is in BSQ format, to one if it is in BIL format, or to two if it is in BIP format.

- **MATCH_ID**
  
  Set this keyword equal to the tile ID of a previously initialized piece of tile data in order to match the tile size of the current data to the size of the previously initialized data. This is used to get consistent tiles for multiple tile requests.

- **NUM_TILES**
  
  Use this keyword to specify a named variable that will contain the number of tiles in the request.

- **OVERLAP**
  
  Use this keyword to specify the pixel overlap wanted for each spatial tile. OVERLAP is only used for INTERLEAVE of zero.
TILE_SCALE

Use this keyword to specify the scaling factor applied to the size of BSQ tiles. This allows the user to convert byte data to float or double without using enormous amounts of memory. A scaling factor of 2.0 reduces the tile size by 2; a scaling factor of 1.0 has no effect. The default is 1.0.

Typically, TILE_SCALE is set to the ratio of the size of the processing data type over the file data type (unless this is less than 1.0, in which case the value is set equal to 1.0).

XS

Use this keyword to specify the X starting index. The default is the first sample.

XE

Use this keyword to specify the X ending index. The default is the last sample.

YS

Use this keyword to specify the Y starting index. The default is the first line.

YE

Use this keyword to specify the Y ending index. The default is the last line.

Example

This example initializes the file processing for FID and the bands in array POS. The value of NUM_TILES is now the total loop count for processes all the tiles.

```
tile_id=envi_init_tile(fid, pos, num_tiles=num_tiles)
```

Notes

ENVI_INIT_TILE must be performed prior to requests for tiled data.

See Also

ENVI_GET_TILE, ENVI_TILE_DONE
ENVI_IO_ERROR

Use this procedure to report Input/Output processing errors.

Calling Sequence

ENVI_IO_ERROR, Proc

Arguments

Proc

A text string to be echoed to the terminal when an I/O error occurs. Generally, Proc should be the name of the procedure in which ENVI_IO_ERROR is called.

Keywords

UNIT (optional)

Use this optional keyword to specify the unit number of the output file being generated. If UNIT is specified and an error occurs, ENVI_IO_ERROR will delete this file.

Example

The following example shows how to display an IO error when MY_ERROR is not equal to zero.

```plaintext
if (my_error ne 0) then $
   envi_io_error, 'An error occurred in my processing.'
```
ENVI_LAYER_STACKING_DOIT

Use this processing routine to build a new multiband file from georeferenced images of various pixel sizes, extents, and projections. The input bands will be resampled and reprojected to a common output projection and pixel size. The output file will contain only data based on the map extent of the input images. The user can select either an inclusive (encompass all the files) or exclusive (encompass file overlap) output image.

**Calling Sequence**

ENVI_DOIT, ‘ENVI_LAYER_STACKING_DOIT’

**Keywords**

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation for each input file. DIMS is an array of long integers, [5, # input files], with the following definitions:

- DIMS[0,i]: Unused for this function, set to -1.
- DIMS[1,i]: The starting X pixel for the ith input layer. (The first pixel is number zero.)
- DIMS[2,i]: The ending X pixel for the ith input layer.
- DIMS[3,i]: The starting Y pixel for the ith input layer. (The first pixel is number zero.)
- DIMS[4,i]: The ending Y pixel for the ith input layer.

**EXCLUSIVE (optional)**

Set this optional keyword to specify an exclusive range, encompassing file overlap, based on the map extent of each input layer. The default is an inclusive range, encompassing all files. The spatial size of a layer stacked image generated with the EXCLUSIVE keyword set will always be equal to or smaller than the inclusive result.

**FID**

Use this keyword to specify the file IDs for the input files. FID is an array of input file IDs with one file ID for each input file (layer). This is the value returned from the
keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1. The number of elements of POS and FID correspond to the number of input files to stack together, one entry for each file.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers with one entry for each input file. The values of POS range from zero to the number of bands-1 of each input file. The number of elements of POS and FID correspond to the number of input files to stack together, one entry for each file.

**IN_MEMORY (optional)**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk and OUT_NAME must be specified. If IN_MEMORY is set then OUT_NAME is not used.

**INTERP (optional)**

Set this keyword equal to one of the following values to specify the resampling method:

- 0 - Nearest Neighbor
- 1 - Bilinear
- 2 - Cubic Convolution

The default is to use Nearest Neighbor, method zero.

**OUT_BNAME (optional)**

Use this optional keyword to specify a string array of output band names.

**OUT_DT**

Set this keyword to specify the IDL data type of the output file, using the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. OUT_NAME is a string variable specifying the filename and path of the output file.

OUT_PROJ

Use this keyword to specify the output projection for the layer stacked file. OUT_PROJ is a projection structure returned from ENVI_GET_PROJECTION or ENVI_PROJ_CREATE.

OUT_PS

Use this keyword to specify the output X and Y pixel size. OUT_PS is a two-element double precision array of the output X and Y pixel sizes, respectively. OUT_PS is in the same units as specified in OUT_PROJ.

R_FID (optional)

Use this optional keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data. An invalid file ID is specified as -1.

Example

This example is used to layer stack the Bighorn TM and DEM images together. The example will use an inclusive range, encompassing all the files, with cubic convolution resampling. Each of the six TM bands and the single DEM band will be output into a new layer stacked image with the projection of the TM data.

This example uses the following files found in the DATA directory of the ENVI installation:

- bhtmref.img
- bhtmref.hdr

and the following files found in the bh_3d directory of the ENVI Data CD #1:

- bhdemsub.img
- bhdemsub.hdr

forward_function envi_get_projection

pro example_envi_layer_stacking_doit
    ;
    ; First restore all the base save files.
    ;
envis /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
;
envi_batch_init, log_file='batch.txt'
;
; Open the first input file.
; We will also open the one band
; dem file to layer stack with
; this file.
;
envi_open_file, 'bhtmref.img', r_fid=t_fid
if (t_fid eq -1) then begin
  envi_batch_exit
  return
endif
;
; Open the second input file.
;
envi_open_file, 'bhdemsunb.img', r_fid=d_fid
if (d_fid eq -1) then begin
  envi_batch_exit
  return
endif
;
; Use all the bands from both files
; and all spatial pixels. First build the
; array of FID, POS and DIMS for both
; files.
;
envi_file_query, t_fid, $
  ns=t_ns, nl=t_nl, nb=t_nb
'envi_file_query, d_fid, $
  ns=d_ns, nl=d_nl, nb=d_nb
;
b = t_nb + d_nb
fid = lonarr(nb)
pos = lonarr(nb)
dims = lonarr(5,nb)
;
for i=0L,t_nb-1 do begin
  fid[i] = t_fid
  pos[i] = i
  dims[0,i] = [-1,0,t_ns-1,0,t_nl-1]
endfor
;
for i=t_nb,nb-1 do begin
  fid[i] = d_fid
pos[i] = i-t_nb
dims[0,i] = [-1,0,d_ns-1,0,d_nl-1]
endfor

; Set the output projection and
; pixel size from the TM file. Save
; the result to disk and use floating
; point output data.
;
out_proj = envi_get_projection(fid=t_fid, $
    pixel_size=out_ps)
out_name = 'testimg'
out_dt = 4

; Call the layer stacking routine. Do not
; set the exclusive keyword allow for an
; inclusive result. Use cubic convolution
; for the interpolation method.
;
envi_doit, 'envi_layer_stacking_doit', $
    fid=fid, pos=pos, dims=dims, $
    out_dt=out_dt, out_name=out_name, $
    interp=2, out_ps=out_ps, $
    out_proj=out_proj, r_fid=r_fid

; Exit ENVI
;
envi_batch_exit
end

See Also

MOSAIC_DOIT, REG_WARP_DOIT,
ENVI_CONVERT_FILE_MAP_PROJECTION
ENVI_MAP_INFO_CREATE

This routine returns an ENVI map information structure for any of the supported map projections (see Appendix D, “ENVI Map Projections” in the ENVI User’s Guide). Arbitrary, Geographic, UTM and State Plane map information structures use their corresponding keyword. All other projections use the PROJ keyword to define the associated projection. Projections can be defined using ENVI_PROJ_CREATE. Using the optional keywords you can define the datum, name, units, projection parameters, pixel size, rotation, and the map coordinate. The result of this routine can be used for functions that require an input MAP_INFO structure. For example, to georeference an image, ENVI_SETUP_HEAD requires an input MAP_INFO structure.

This routine should be used instead of accessing the map information structure directly.

Calling Sequence

Result = ENVI_MAP_INFO_CREATE()

Keywords

ARBITRARY (optional)

Set this keyword to specify that an Arbitrary projection for the map information is created. Set the keyword MAP_BASED to create a map based projection. The default is non map based projection. A map-based projection uses the lower left corner of the image as the projection origin, while a pixel-based projection (non-map based) uses the upper left corner as the origin.

DATUM (optional)

Use this keyword to specify the datum for the map information projection. The default for Geographic is WGS-84 and the default for UTM and State Plane is North America 1927. All other projections default to no datum. The exact name that ENVI uses for each datum is listed in the datum.txt file in the map_proj directory of your ENVI installation.

GEOGRAPHIC (optional)

Set this keyword to specify that a Geographic projection for the map information is created.
**MAP_BASED (optional)**

Set this keyword to specify that the Arbitrary projection should be interpreted as map based. This keyword does not have any effect for other projections.

**MC**

Set this keyword to specify the map location tie point. The tie point is the reference point for a pixel at a known map coordinate. MC is a four elements double precision array where

- \( mc(0) \) is the X pixel location corresponding to the X map location, \( mc(2) \)
- \( mc(1) \) is the Y pixel location corresponding to the Y map location, \( mc(3) \)
- \( mc(2) \) is the X map location corresponding to the X pixel location, \( mc(0) \)
- \( mc(3) \) is the Y map location corresponding to the Y pixel location, \( mc(1) \)

**NAME (optional)**

Use this keyword to specify the name of the map information projection. NAME is a string variable

**PARAMS (optional)**

Use this keyword to specify the parameters for the map information projection. PARAMS is a double array with 1 to 15 elements. The number of elements of PARAMS is determined by the projection type (see the TYPE keyword). PARAMS is not used with Arbitrary, Geographic, State Plane, or UTM projections. See the ENVI Map Projections Appendix in the Users Guide for a full list of the projection type and their corresponding projection parameters. The PARAMS keyword must contain all projection parameters listed in the ENVI Map Projections Appendix (for the specified projection) except the datum and name. The datum and name are specified using the DATUM and NAME keywords, respectively.

**PROJ (optional)**

Set this optional keyword to specify the projection for the map information. PROJ is an ENVI projection structure returned from ENVI_PROJ_CREATE or ENVI_GET_PROJECTION. PROJ is not needed if the keyword ARBITRARY, GEOGRAPHIC, STATE_PLANE, or UTM is used. For all other projections, either set the PROJ keyword or define the projection using the keywords DATUM, NAME, PARAMS, SOUTH, TYPE, UNITS, and ZONE.
PS

Set this keyword to specify the pixel size of the image. PS is a two element double precision array where

- \( ps(0) \) is the X pixel size
- \( ps(1) \) is the Y pixel size

ROTATION (optional)

Set this keyword to specify the rotation of the image in the defined projection. Rotation is expressed in degrees clockwise from North.

SOUTH (optional)

Set this keyword to specify that the UTM projection is in the southern hemisphere.

STATE_PLANE (optional)

Set this keyword to specify that a State Plane projection for the map information is created.

TYPE (optional)

Use this keyword to specify the map information projection type. TYPE is an integer value corresponding to the projection type. See the ENVI Map Projections Appendix in the Users Guide for a full list of the projection types and their corresponding projection value.

UNITS (optional)

Use this keyword to specify the projection units. UNITS is an integer value indicating the projection units. The function \texttt{ENVI\_TRANSLATE\_PROJECTION\_UNITS} will convert projection unit strings to integer values. The default UNITS are Degrees for Geographic, Feet for State Plane and Meters for all other projections.

UTM (optional)

Set this keyword to specify that a UTM projection for the map information is created.

ZONE (optional)

Use this keyword to specify the UTM or State Plane zone number. ZONE is only used for UTM and State Plane projections.
Example

This example creates a Geographic map information structure with a Geographic projection, default datum of WGS-84, default units of Degrees, and a pixel size of one second (1./3600). Set the map tie point for the center of the first pixel (.5,.5) to 34.5 degrees North and 117.4 degrees West. This is one of the simplest uses of ENVI_MAP_INFO_CREATE.

```
; Set the pixel size and map tie point
; ps = [1D/3600, 1D/3600]
mc = [0D, 0, -117.4, 34.5]
; Create the map information
map_info = envi_map_info_create(/geographic, $ 
   mc=mc, ps=ps)
```

This example creates map information for a UTM projection zone 23 South with units of kilometers and a North America 1983 datum. Set the map tie point for the upper left corner of the first pixel to 8339330 North and 177246 East. We use the keywords to ENVI_MAP_INFO_CREATE instead of first creating a projection structure and then using the PROJ keyword.

```
; First convert the kilometers to its integer representation
; using ENVI_TRANSLATE_PROJECTION_UNITS
; units = ENVI_TRANSLATE_PROJECTION_UNITS('km')
; Set the datum and map tie points
; datum = 'North America 1983'
mc = [0D, 0, 177246, 8339330]
ps=[30, 30]
; Now create the UTM map information
map_info = ENVI_MAP_INFO_CREATE(/UTM, ZONE=23, /SOUT, $ 
   DATUM = datum, UNITS = units, MC = mc, PS = ps)
```
See Also

ENVI_PROJ_CREATE,
ENVI_TRANSLATE_PROJECTION_UNITS,ENVI_TRANSLATE_PROJECTION_NAME,ENVI_CONVERT_PROJECTION_COORDINATES
ENVI MASK APPLY_DOIT

Use this program to apply a mask to a file.

Calling Sequence

ENVI_DOIT, ‘ENVI_MASK_APPLY_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

M_FID

Use this keyword to specify the file ID for the mask file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

M_POS

Use this keyword to specify the band position of the mask band. M_POS is a single long value greater than or equal to zero.
OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify a one-dimension array of band positions indicating the band numbers to perform the operations on. POS is a long array ranging from 0 to the number of bands-1.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

VALUE
Use this keyword to specify the mask value. Each masked pixel in the output image will take the value VALUE.

Example

```plaintext
pro m_apply_mask_1

; Initialize ENVI
get_lun, lunit
openw, lunit, '/batch/b.log'
envi_batch_init, /batch_mode, batch_unit=lunit

; Set the input and output file names
out_name = '/data/batch_test/out_file'
envi_open_file, '/data/batch_test/in.bil', r_fid=fid
environ_open_file, '/data/batch_test/in_file', r_fid=m_fid
if (fid eq -1 or m_fid eq -1) then begin
  free_lun, lunit
  envi_batch_exit
  return
endif

; get some useful information and set the output filename.
envi_file_query, fid, ns=ns, nl=nl, nb=nb, bname=bname
```
; Set the keyword parameters
dims = [-1L, 0, ns-1, 0, nl-1]  
pos = lindgen(nb)  
m_pos = [0]

; Call the 'doit'
envi_mask_apply_doit, fid=fid, pos=pos, dims=dims, m_fid=m_fid,$
  m_pos=m_pos, value=0, out_name=out_name, in_memory=0, $
  r_fid=r_fid

; Exit ENVI
envi_batch_exit
free_lun, lunit
end
ENVI_NEURAL_NET_DOIT

Use this program to perform classification using a Neural Net method. Neural Net classification is a supervised classification where the net is trained on a set of input regions of interest (ROIs). The keywords ALPHA, ETA, THETA, NUM_SWEEPS, NUM_LAYERS and RMS_CRIT are associated with training of the net. The optional keyword THRESH can be used to set the minimum activation threshold a class must satisfy in order to classified.

Calling Sequence

ENVI_DOIT, ‘ENVI_NEURAL_NET_DOIT’

Keywords

ALPHA

Use this keyword to specify the training momentum. ALPHA is a floating point or double precision number between zero and one.

ACT_TYPE

Use this keyword to specify the type of activation function for training the neural net. ACT_TYPE is an integer value set to one of the following values:

- 0 - Logistic
- 1 - Hyperbolic

CLASS_NAMES

Use this keyword to specify names for each output class. CLASS_NAMES is an array of strings with num_classes+1 elements. Remember to set the zero class to “Unclassified”.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
DIMS(3): The starting Y pixel. (The first pixel is number zero.)
DIMS(4): The ending Y pixel.

ETA
Use this keyword to specify the training rate. ETA is a floating point or double precision number between zero and one.

FID
Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY
Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

LOOKUP
Use this keyword to specify an array specifying the color tables for the classification image. Each output class can have a unique color triple [r,g,b], LOOKUP is a byte array of size (3, num_classes+1). Remember that class zero must also have a color triplet (commonly black [0,0,0]).

NUM_CLASSES
Use this keyword to specify the number of output classes. The number of output classes is equal to the number of input regions of interest (ROIs) as specified by ROI_PTR.

NUM LAYERS
Use this keyword to specify the number of hidden layers in the neural net classifier. Typically this value is set to zero, one or two.

NUM_SWEEPS
Use this keyword to specify the maximum number of training sweeps to perform count. The actual number of training sweeps may be less than NUM_SWEEPS if error criteria has been met, see RMS_CRIT.

OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID (optional)

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

RMS_CRIT

Use this keyword to specify the RMS training error criteria. During training if the RMS error is less than RMS_CRIT then the training ended and the image is classified according to the trained neural net.

ROI_ID

Use this keyword to specify an array of ROI IDs returned from a call to ENVI_GET_ROI_IDS. Each ID in the array will use the corresponding ROI to calculate both a co-polarization and cross-polarization image.

RULE_FID (optional)

Use this keyword to specify a named variable that will contain the file ID for the processed rule image. This file ID can be used to access the processed data.

RULE_OUT_BNAME (optional)

Use this keyword to specify a string array that contains the output band names for the rule image.

RULE_OUT_NAME (optional)

Use this keyword to specify an output filename for the rule image. If this item is present the rule image is automatically saved.

RULE_IN_MEMORY (optional)

Set this keyword to specify that output rule images should be stored in memory.
**THETA**

Use this keyword to specify the training threshold contribution. THETA is a floating point or double precision number between zero and one.

**THRESH (optional)**

Use this keyword to specify an minimum activation threshold a class must have in order to be classified.

**TRAIN**

Set this keyword to train the neural net prior to classification. Currently this keyword must be set.

**Example**

```plaintext
forward_function envi_get_roi_ids, envi_get_roi_dims_ptr
pro example_envi_neural_net_doit
    ; First restore all the base save files.
    ; envi, /restore_base_save_files
    ; Initialize ENVI and send all errors and warnings to the file batch.txt
    ; envi_batch_init, log_file='batch.txt'
    ; Open the input file
    ; envi_open_file, 'bhtm_mnf.img', r_fid=fid
    if (fid eq -1) then begin
        envi_batch_exit
        return
    endif
    ; Set the dims and pos to classify all data (spatially and spectrally) in the file.
    ; envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1L, 0, ns-1, 0, nl-1]  
pos = lindgen(nb)
out_name = 'testimg'
    ; Restore the ROI file used as the training pixels. Each ROI in the file
```
; will be considered on input class
;
envi_restore_rois, 'bhtm_nd.roi'
roi_ids = envi_get_roi_ids(fid=fid, $
   roi_colors=lookup, roi_names=class_names)
;
; Specify the neural net training
; criteria.
;
theta = .9
eta = .2
alpha = .9
act_type = 0
rms_crit = .1
num_layers = 3
num_sweeps = 1000
;
; Set the classification variables
;
num_classes = n_elements(roi_ids)
class_names = ['Unclassified', class_names]
lookup = reform([0,0,0, $
   reform(lookup,3*num_classes)],3,num_classes+1)
;
; Call the doit
;
envi_doit, 'envi_neural_net_doit', $
   fid=fid, pos=pos, dims=dims, $
   out_name=out_name, rule_out_name='', $
   theta=theta, eta=eta, alpha=alpha, $
   num_classes=num_classes, num_sweeps=num_sweeps, $
   num_layers=num_layers, act_type=act_type, $
   rms_crit=rms_crit, roi_ids=roi_ids, /train, $
   class_names=class_names, lookup=lookup
;
; Exit ENVI
;
envi_batch_exit
end

See Also

CLASS_DOIT, CLASS_RULE_DOIT, CLASS_STATS_DOIT
ENVI_OPEN_DATA_FILE

Use this routine to open a data file. The types of files opened include ADRG, AVHRR, BMP, DMSP, DOQ, ENVI, EOSAT IRS and TM, Erdas 7.5 and 8.x, ErMapper, ERS, ESA Sharp, ERA Landsat TM, HDF, JERS, MAS50, MRLC, NITF, NLAPS, PCI, RADARSAT, SeaWiFS, SPOT, Tiff, and TopSAR.

Calling Sequence

ENVI_OPEN_DATA_FILE, Name

Arguments

NAME

The filename to open. The type of the file is indicated by setting the proper keyword.

Keywords

ADRG (optional)

Set this keyword to specify that the file being opened is a ADRG file.

AVHRR (optional)

Set this keyword to specify that the file being opened is a AVHRR file.

BMP (optional)

Set this keyword to specify that the file being opened is a BMP file.

DMSP (optional)

Set this keyword to specify that the file being opened is a DMSP file.

DOQ (optional)

Set this keyword to specify that the file being opened is a DOQ file.

ENVI (optional)

Set this keyword to specify that the file being opened is an ENVI file.

EOSAT_IRS (optional)

Set this keyword to specify that the file being opened is an EOSAT IRS file.
EOSAT_TM (optional)
Set this keyword to specify that the file being opened is an EOSAT TM file.

ERDAS75 (optional)
Set this keyword to specify that the file being opened is an Erdas 7.5 file.

ERDAS80 (optional)
Set this keyword to specify that the file being opened is an Erdas 8.x file.

ERMAPPER (optional)
Set this keyword to specify that the file being opened is an ERMapper file.

ERS (optional)
Set this keyword to specify that the file being opened is an ERS file.

ESA_SHARP (optional)
Set this keyword to specify that the file being opened is an ESA Sharp file.

ESA_TM (optional)
Set this keyword to specify that the file being opened is an ESA Landsat TM file.

HDF_SD (optional)
Set this keyword to specify that the file being opened is a HD file containing SD data.

JERS (optional)
Set this keyword to specify that the file being opened is a Jers file.

MAS_50 (optional)
Set this keyword to specify that the file being opened is a MAS_50 file.

MRLC (optional)
Set this keyword to specify that the file being opened is a MRLC file.

NITF (optional)
Set this keyword to specify that the file being opened is a NITF file.
**NLAPS (optional)**
Set this keyword to specify that the file being opened is a NLAPS file.

**PCI (optional)**
Set this keyword to specify that the file being opened is a PCI file.

**RADARSAT (optional)**
Set this keyword to specify that the file being opened is a RADARSAT file.

**R_FID**
Use this keyword to specify a named variable that will contain the file ID for the opened data. This file ID can be used to access the processed data.

**SEAWIFS (optional)**
Set this keyword to specify that the file being opened is a SeaWiFS file.

**SPOT (optional)**
Set this keyword to specify that the file being opened is a SPOT file.

**TIFF (optional)**
Set this keyword to specify that the file being opened is a TIFF or GEOTIFF file.

**TOPSAR (optional)**
Set this keyword to specify that the file being opened is a TOPSAR file.

**Example**

```plaintext
pro example_envi_open_data_file

; Open a data file
fname = 'lon_spot'
envi_open_data_file, fname, /ermapper, r_fid=fid
if (fid eq -1) then return

; Query and print the ns, nl, and nb
envi_file_query, fid, ns=ns, nl=nl, nb=nb
print, ns, nl, nb
end
```
ENVI_OPEN_FILE

Use this procedure to open an ENVI file. The value of R_FID will be the reference for accessing any information about this file.

Calling Sequence

ENVI_OPEN_FILE, fname

Arguments

Fname

The file name, including the path, of the file to open.

Keywords

NO_REALIZE (optional)

Set this optional keyword to suppress opening of the Available Bands List. If the Available Bands List is already open this keyword has no effect.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the opened file. R_FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

Example

Use ENVI_OPEN_FILE to open a file. Check and make sure the file was opened properly before using the returned file ID to perform an ENVI_FILE_QUERY.

; Input file definition
fname = '/data/img_001'
envi_open_file, fname, r_fid=fid

if (fid eq -1) then return
envis_file_query, fid, ns=ns, nl=nl, nb=nb

See Also

ENVI_FILE_MNG, ENVI_FILE_QUERY, ENVI_PICKFILE
ENVI_OUTPUT_TO_EXTERNAL_FORMAT

Use this program to output image data to external formats. This routine allows output of ArcView, ASCII, ENVI, ErMapper, Erdas, PCI and TIFF formats.

Calling Sequence

ENVI_OUTPUT_TO_EXTERNAL_FORMAT

Keywords

ARCVIEW (optional)
Set this keyword to specify output to a ArcView formatted file.

ASCII (optional)
Set this keyword to specify output to a ASCII formatted file. If the ASCII keyword is set you must also specify the FIELD keyword.

DIMS
Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

ENVI (optional)
Set this keyword to specify output to an ENVI formatted file. If the ENVI keyword is set you may optionally specify the band names using the OUT_BNAME keyword.

ERDAS (optional)
Set this keyword to specify output to an Erdas .lan formatted file.

ERMAPPER (optional)
Set this keyword to specify output to an ErMapper formatted file.
**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**FIELD (optional)**

Use this keyword to specify a two element long array of the ASCII character field width and precision. The first element of the array specifies the number of characters in the external field. The second element in the array specifies the number of positions after the decimal point.

**OUT_BNAME (optional)**

Use this keyword to specify a string array of output band names, if desired. This keyword is only valid when the ENVI keyword is also set.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. The output file will be in the format specified by one of the following keywords, ARCVIEW, ASCII, ENVI, ERMAPPER, ERDAS, PCI or TIFF.

**PCI (optional)**

Set this keyword to specify output to a PCI formatted file.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

**TIFF (optional)**

Set this keyword to specify output to a TIFF formatted file.
ENVI_PICKFILE

Use this function to pick a filename. This is different than ENVI_SELECT which selects an open file or band.

Calling Sequence

Result = ENVI_PICKFILE()

Keywords

DEFAULT (optional)
Use this optional keyword to specify a default filename and output path.

DIRECTORY (optional)
Set this optional keyword to allow selection of an output directory instead of a file.

FILTER (optional)
Use this optional keyword to specify the filename filter to apply to the file list.

NO_CHANGE (optional)
Set this optional keyword to inhibit the changing of the current ENVI output or data directory regardless of what directory the file was selected from. Normally the directory is updated to the location of the selected file.

OUTPUT (optional)
Set this optional keyword in order to use the current ENVI output directory instead of the current ENVI data directory.

TITLE
Use this keyword to specify the title of the file selection widget.

Example

Use ENVI_PICKFILE to select a filename, by default only list the files that have a .txt extension.

    name = envi_pickfile(title='Pick a text file', $
                        filter='*.txt')
    if (name eq '') then return
    print, 'The selected filename is: ', name
See Also

ENVI_OPEN_FILE, ENVI_SELECT
ENVI_PLOT_DATA

Use this procedure to display an XY plot(s) in a new ENVI plot window. Depending on the dimensions of the Y array one or more plots will be displayed.

Calling Sequence

ENVI_PLOT_DATA, X, Y

Arguments

X
An array of X values to plot. X is an array of N points.

Y
An array of Y values to plot. Y is an array of \((N, \text{number of plots})\).

Keywords

BASE (optional)
Use this optional keyword to specify a named variable that will contain the returned widget base of the plot window.

GROUP (optional)
Use this optional keyword to specify the group leader for this widget. The default is the ENVI main base which will remove the plot window when ENVI is exited.

TITLE (optional)
Use this optional keyword to specify the plot window title. The default is ENVI Plot Window.

PLOT_COLORS (optional)
Use this optional keyword to specify the plot graphic color index. PLOT_COLORS is a long array of \((\text{number of plots})\), specifying graphic color index.

PLOT_NAMES (optional)
Use this keyword to specify the plot names. PLOT_NAMES is a string array of \((\text{number of plots})\).
PLOT_STYLES (optional)

Use this optional keyword to specify the plot styles. PLOT_STYLES is long array of (number of plots), specifying the style index. The available plot style indexes are as follows:

- 0  Solid
- 1  Dotted
- 2  Dashed
- 3  Dash Dot
- 4  Dash Dot Dot Dot
- 5  Long Dashes

PLOT_TITLE (optional)

Use this optional keyword to specify the plot title. PLOT_TITLE is a string.

XTITLE (optional)

Use this optional keyword to specify the X axis title.

YTITLE (optional)

Use this optional keyword to specify the Y axis title.

Example

The example plot creates a plot window with two plots.

```plaintext
x = findgen(100)
y = reform([x, sin(x)], 100, 2)
plot_names = ['XY Line', 'SIN(X)']
envi_plot_data, x, y, plot_names=plot_names
```

See Also

ENVI_READ_COLS
ENVI_PROJ_CREATE

Use this routine to create an ENVI projection structure for any of the supported projections, see Appendix D, “ENVI Map Projections” in the ENVI User’s Guide. Using the optional keywords you can define the datum, name, units, and projection parameters. Additional keywords allow the definition with common defaults for the Arbitrary, Geographic, State Plane and UTM projections. This routine should be used instead of accessing the projection structure directly.

Calling Sequence

Result = ENVI_PROJ_CREATE( )

Keywords

ARBITRARY (optional)

Set this keyword to specify that an Arbitrary projection is created. Set the keyword MAP_BASED to create a map based projection. The default is non map based projection. A map-based projection uses the lower left corner of the image as the projection origin, while a pixel-based projection (non-map based) uses the upper left corner as the origin.

DATUM (optional)

Use this keyword to specify the datum for the projection. The default for Geographic is WGS-84 and the default for UTM and State Plane is North America 1927. All other projections default to no datum. The exact name that ENVI uses for each datum is listed in the datum.txt file in the map_proj directory of the ENVI installation.

GEOGRAPHIC (optional)

Set this keyword to specify that a Geographic projection is created.

MAP_BASED (optional)

Set this keyword to specify that the Arbitrary projection should be interpreted as map based. This keyword does not have any effect for other projections.

NAME (optional)

Use this keyword to specify a user defined name for the projection. NAME is a string variable. The actual projection type is not determined by the value of NAME—
instead the projection type is determined by the keyword TYPE, ARBITRARY,
GEOGRAPHIC, STATE_PLANE, or UTM.

**PARAMS (optional)**

Use this keyword to specify the parameters for the projection. PARAMS is a double
array with 1 to 15 elements. The number of elements of PARAMS is determined by
the projection type (see the TYPE keyword).

PARAMS is not used with Arbitrary, Geographic, State Plane, or UTM projections.
See Appendix D, “ENVI Map Projections” in the ENVI User’s Guide for a full list of
the projection types and their corresponding projection parameters. The PARAMS
keyword must contain all projection parameters listed in the ENVI Map Projections
Appendix (for the specified projection) except the datum and name. See the example
below for an explanation of some common projection parameters. The datum and
name are specified using the DATUM and NAME keywords, respectively.

**SOUTH (optional)**

Set this keyword to specify that the UTM projection is in the southern hemisphere.

**STATE_PLANE (optional)**

Set this keyword to specify that a State Plane projection is created.

**TYPE (optional)**

Use this keyword to specify the projection type. TYPE is an integer value
corresponding to the projection type. See the ENVI Map Projections Appendix in the
Users Guide for a full list of the projection types and their corresponding projection
value.

**UNITS (optional)**

Use this keyword to specify the projection units. UNITS is an integer value indicating
the projection units. The function ENVI_TRANSLATE_PROJECTION_UNITS will
convert projection unit strings to integer values. The default UNITS are Degrees for
Geographic, Feet for State Plane and Meters for all other projections.

**UTM (optional)**

Set this keyword to specify that a UTM projection is created.

**ZONE (optional)**

Use this keyword to specify the UTM or State Plane zone number. ZONE is only
used for UTM and State Plane projections.
Example

This example creates a geographic projection with default values for the datum (WGS-84), name (Geographic Lat/Lon), and the units (Degrees). This is one of the simplest uses of `ENVI_PROJ_CREATE`.

```c
proj = ENVI_PROJ_CREATE(/geographic)
```

This example creates a UTM projection for zone 23 South using the North America 1983 datum. The projection will have units of kilometers.

```c
; First convert the kilometers to its integer representation
; using ENVI_TRANSLATE_PROJECTION_UNITS and set
; the datum
;
units = ENVI_TRANSLATE_PROJECTION_UNITS('km')
datum = 'North America 1983'
;
; Now create the UTM projection
;
Proj = ENVI_PROJ_CREATE(/utm, zone=23, /south, $
    datum=datum, units=units)
```

This example creates a Transverse Mercator projection. Using the information in Appendix D, “ENVI Map Projections” of the *ENVI User’s Guide*, we see that this projection, ENVI projection type “3”, requires the following parameter:

```c
a, b, lat0, lon0, x0, y0, and k0
```

where

- `a` - the equatorial radius (semi-major axis)
- `b` - the polar radius (semi-minor axis)
- `lat0` - Latitude of origin of projection
- `lon0` - Longitude of central meridian
- `x0` - False easting
- `y0` - False Northing
- `k0` - Scale factor at central meridian

The actual values for the parameters are based on the projection being defined. In this example, the datum uses the Australian National ellipsoid which provides the values for `a` and `b`. The Latitude of Origin is 0.0 degrees and the Longitude of Central Meridian is 99 degrees. The false Northing and Easting are 10000000 and 50000, respectively. The Scale factor is set to .9996. In addition to these parameters, this projection will use the Australian Geodetic 1966 datum and have the name Australian Map Grid (AGD 66) Zone 47. We will also use the default of meters for the map units.
; Define the PARAMS values
;
Params = [6378160.0, 6356774.7, $0.000000, 99.000000, $500000., 1000000., $ .9996]
;
; Define the Datum and projection name
datum = 'Australian Geodetic 1966'
name = 'Australian Map Grid (AGD 66) Zone 47'
;
; Create the projection
;
proj = ENVI_PROJ_CREATE(type=3, $
   name=name, datum=datum, params=params)

See Also

ENVI_TRANSLATE_PROJECTION_UNITS,
ENVI_TRANSLATE_PROJECTION_NAME, ENVI_CONVERT_PROJECTION_COORDINATES, ENVI_MAP_INFO_CREATE
ENVI_READ_COLS

Use this procedure to read ASCII column data.

Calling Sequence

ENVI_READ_COLS, Name, Values

Arguments

Name

The name of the file to read from.

Values

A named variable containing the values read from file.

Keywords

ERROR

Use this keyword to specify a named variable that will hold the value of the error flag. If this variable is set equal to one, there was a read error. A value of zero indicates no error.

HEAD

Use this keyword to specify a string array that will contain the column headers of the file being read.

READ_HEAD

Set this keyword to read the column header and store the result in the string array specified by the keyword HEAD.

READ_SKIP (optional)

Set this optional keyword to saves all skipped lines in the array specified by the SKIP keyword. Lines are only skipped when they cannot be decoded into the proper values.

SKIP

Use this keyword to specify a string array that will contain any skipped lines. The keyword must be specified if READ_SKIP is set.
ENVI_REPORT_INC

Use this procedure to set the increment used for tiled processing. This value is shown in ENVI status windows as a value in the text box labeled “Inc:”

**Calling Sequence**

ENVI_REPORT_INC, Base, Num_tiles

**Arguments**

**Base**

The base ID of the status window widget.

**Num_tiles**

The total number of processing cycles. Typically this is equal to the number of tiles.

**Example**

ENVI_REPORT_INC should be called before the actual processing loop begins:

```
envi_report_inc, base, num_tiles
for i=0,num_tiles-1 do begin
   envi_report_stat, base, i, num_tiles
   .
   .
   .
   algorithm here
   .
   .
endfor
```

**See Also**

ENVI_REPORT_STAT, ENVI_REPORT_INIT
ENVI_REPORT_INIT

This procedure displays an ENVI status reporting box showing the percentage of the function completed and the increment in use. It also provides a Cancel button for the function. This function is called to initialized the status window and again after the processing is finished.

![Image of status window]

Figure 9-1: Processing status dialog.

Calling Sequence

ENVI_REPORT_INIT, Rstr

Arguments

Rstr

An array of strings to display in the status window. Each element in the array is displayed on a new line.

Keywords

BASE

Use this keyword to specify a named variable that will contain the widget base used to display the status window. The base value returned from initialization must be supplied to ENVI_REPORT_INIT when removing the status window.
**FINISH**

Set this keyword to remove the status window after processing has finished. Note that the BASE keyword must be set equal to the same base value as was specified when the window was created.

**INTERUPT**

Set this keyword to allow processing interrupts using the Cancel button.

**TITLE**

Set this keyword equal to the string to be displayed in the status window’s title bar.

**Example**

The following commands would create a window like the one shown above.

```plaintext
if(in_memory) then ostr = 'Output to Memory' $
else ostr = 'Output File: ' + out_fname
rstr = ["Input File :" + fname, ostr]
envi_report_init, rstr, title="USGS Munsell", base=base
```

This function also must be called at the end of processing to remove the report widget:

```plaintext
envi_report_init, base=base, /finish
```

**See Also**

`ENVI_REPORT_STAT`, `ENVI_REPORT_INC`
ENVI_REPORT_STAT

This procedure updates the processing status percent completed as tile processing occurs. Each time this procedure is called the widget is updated based on the values of the arguments Num and Den.

Calling Sequence

ENVI_REPORT_STAT, Base, Num, Den

Arguments

Base
The base ID of the status window widget to be updated. This is the value returned from the ENVI_REPORT_INIT at initialization

Num
A variable which serves as a counter for the number of tiles processed. The percent completed is determined by the ratio of Num over Den.

Den
The total number of tiles to be processed. The percent completed is determined by the ratio of Num over Den.

Keywords

CANCEL
Use this keyword to specify a named variable that returns the status of the cancel button. A returned value of “1” indicates the cancel button was pressed, “0” is returned otherwise. This keyword only has meaning if INTERRUPT was specified in ENVI_REPORT_INIT.
Example

This example shows how to update the percentage completed for an ENVI status report window created with ENVI_REPORT_INIT.

```plaintext
for i=0, num_tiles-1 do begin
    envi_report_stat, base, i, num_tiles
    .
    the user function and tiling is done here
    .
endfor
```

Notes

The parameters *Num* and *Den* are used to form the ratio \((\text{Num}/\text{Den}) \times 100\) for calculation of the percent complete. In the example above, the percent complete would be calculated as \((i/\text{num\_tiles})\times100\).

See Also

ENVI_REPORT_INIT, ENVI_REPORT_INC
**ENVI_RESTORE_ROIS**

This procedure is used to restore a previously saved Region of Interest (ROI) file.

**Calling Sequence**

```
ENVI_RESTORE_ROIS, Fname
```

**Arguments**

- **Fname**
  
The filename of the saved ROI file.

**Example**

```
Restore the ROI file wetlands.roi saved in the directory d:\data\tm.

   envi_restore_rois, 'd:\data\tm\wetlands.roi'
```

**See Also**

- `ENVI_GET_ROI`, `ENVI_GET_ROI_DIMS_PTR`, `ENVI_GET_ROI_IDS`
ENVI_ROI_TO_IMAGE_DOIT

Use this program to create a classification image from ROIs. Each input ROI represents a class in the output image.

Calling Sequence

ENVI_DOIT, ‘ENVI_ROI_TO_IMAGE_DOIT’

Keywords

CLASS_VALUES

Set this keyword to specify the values for the output classes. CLASS_VALUES is a long array specifying the class value for the corresponding ROI_IDS. The class value of zero is reserved for the Unclassified class.

FID

Use this keyword to specify the file ID associated with the ROIs. FID is used to get the number of samples and number of lines for the output file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY (optional)

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

R_FID (optional)

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

ROI_IDS

Use this keyword to specify the ROI Ids to turn into classes. Each ROI takes on the class values specified by CLASS_VALUES, the value of zero is reserved for the unclassified class. ROI_ID is an array of ROI Ids returned from the function ENVI_GET_ROI_IDS.
Example

forward_function envi_get_roi_ids

pro example_envi_roi_to_image_doit

; ; First restore all the base save files.
; envi, /restore_base_save_files
; ; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; ; Open the input file associated with
; the ROIs
; envi_open_file, 'bhtm_mnf.img', r_fid=fid
if (fid eq -1) then begin
   envi_batch_exit
   return
endif
; ; Restore the ROI file and get all
; the available ROI ids.
; envi_restore_rois, 'bhtm_nd.roi'
roi_ids = envi_get_roi_ids()
if (roi_ids[0] eq -1) then return
; ; Set the necessary variables
; out_name = 'testimg'
class_values = lindgen(n_elements(roi_ids))+1
; ; Call the doit
; envi_doit, 'envi_roi_to_image_doit', $
   fid=fid, roi_ids=roi_ids, out_name=out_name, $
   class_values=class_values
; ; Exit ENVI
; envi_batch_exit
end
ENVI_SAVE_ROIS

Use this program to save ROIs from within ENVI. The program accepts the output ROI filename and a list of ROIs to save.

Calling Sequence

ENVI_SAVE_ROIS, Filename, Roi_ids

Arguments

Filename
The output filename for the saved ROIs.

ROI_ids
The ROI IDs to save to the output Filename. This is the value returned from ENVI_GET_ROI_IDS or ENVI_CREATE_ROI.

Example

roi_ids = envi_get_roi_ids()
envi_save_rois, 'test.roi', roi_ids
**ENVI_SEGMENT_DOIT**

Use this program to segment a classification image into unique spatially contiguous “blobs.” Blobs can be created with a minimum population and either four or eight neighbors.

**Calling Sequence**

ENVI_DOIT, ‘ENVI_SEGMENT_DOIT’

**Keywords**

**ALL_NEIGHBORS**

Use this keyword to specify the connectivity for segmentation blobs. If set to one all eight surrounding neighbors are used to connect elements of a blob otherwise set this keyword to zero to specify that only four (up, down, left, right) pixels form connections.

**CLASS_PTR**

Use this keyword to specify the classes to segment. CLASS_PTR is a long array of class numbers ranging from zero (Unclassified) to the number of classes.

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.
**IN_MEMORY (optional)**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**MIN_POPULATION (optional)**

Use this keyword to specify the minimum number of pixels that form a segmentation blob. The default value for MIN_POPULATION is zero.

**OUT_BNAME (optional)**

Use this keyword to specify a string array of output band names, if desired.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

**R_FID (optional)**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**Example**

```plaintext
pro example_envi_segment_doit

; First restore all the base save files.
; envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'

; Open the input file
; envi_open_file, 'bhtm_sam.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
```
return
endif

; Set the keywords. We will perform
; the segmentation on all the entire
; spatial image but segment only
; the first class (the Unclassified
; class is the zero class). There must
; at least 50 pixels in a segment
; are connected with any of the
; eight neighbors.
;
    envi_file_query, fid, ns=ns, nl=nl
dims = [-1, 0, ns-1, 0, nl-1]
    pos = [0]
    out_name = 'testimg'
    min_population = 50
    class_ptr = [1]

; Perform the segmentation.
;
    envi_doit, 'envi_segment_doit', 
        fid=fid, pos=pos, dims=dims, 
        min_population=min_population, 
        class_ptr=class_ptr, 
        /all_neighbors, out_name=out_name
;
; Exit ENVI
;
    envi_batch_exit
end

See Also

CLASS_MAJORITY_DOIT, CLASS_CS_DOIT
ENVI_SELECT

This procedure provides the standard ENVI File Selection dialog for selecting data files open in the current ENVI session. In its default form the dialog allows the selection of a file or band with an option for spatial and spectral subsetting. However, the dialog has a number of keywords that allow precise control over the options allowed during the data selection.

Once the dialog is started, the remaining ENVI session is modal until the “OK” or “Cancel” button is selected. If the “OK” button is selected the value returned in the FID is the file ID of the selected data. This FID value is used in subsequent ENVI routines and is the link to the selected file.

**Figure 9-2: A File Selection dialog.**

**Calling Sequence**

ENVI_SELECT
Keywords

**BAND_ONLY (optional)**

Set this optional keyword to disable selection of files using the “Input File” toggle button and to only allow band selection.

**DIMS (optional)**

Use this optional keyword to specify a named variable that will contain a five-element array holding the data dimensions. The elements are defined as follows:

- DIMS(0): a pointer to the ROI (set to “-1” if no ROI is selected).
- DIMS(1): the starting X coordinate.
- DIMS(2): the ending X coordinate.
- DIMS(3): the starting Y coordinate.
- DIMS(4): the ending Y coordinate.

**FID (optional)**

Use this keyword to specify a named variable that will contain the file ID of the selected file when the “OK” button is selected. If FID[0] = -1, the “Cancel” button was selected and the user should take the appropriate action.

**FILE_ONLY (optional)**

Set this optional keyword to disable selection of single bands using the “Input band” toggle button and only allow file selection.

**FILE_TYPE (optional)**

Set this optional keyword to an integer value specifying the allowable file type. Only files with matching types are considered valid. See “ENVI_FILE_TYPE” on page 417 for details on how to determine the integer value of a file type.

**MASK (optional)**

Set this optional keyword to allow selection of a mask.

**M_FID (optional)**

Use this optional keyword to specify a named variable that will contain the file ID of the selected mask file. To enable mask selection the keyword MASK must be set.
M_POS (optional)
Use this optional keyword to specify a named variable that will contain the band position of the selected mask. To enable mask selection the keyword MASK must be set.

NO_DIMS (optional)
Set this optional keyword to disable spatial subsetting.

NO_SPEC (optional)
Set this optional keyword to disable spectral subsetting.

POS (optional)
Use this optional keyword to specify a named variable that will contain an array holding the bands selected.

ROI (optional)
Set this optional keyword to allow Region of Interest (ROI) subsetting. If selection is subset by ROI then DIMS(0) contains the roi pointer.

TITLE (optional)
Use this optional keyword to specify the string used in the title bar of the File Selection dialog window. Enter the title enclosed in single quotes.

Example
This example shows how to select a file or band and return the FID, DIMS, and POS information. This is typically the simplest use of ENVI_SELECT. In addition, if the “Cancel” button is selected (e.g., FID[0] = -1), then return from the current user procedure.

ENVI_SELECT, fid=fid,dims=dims,pos=pos
if (fid[0] eq -1) then return

The following example will allow selection of a single classification band with no spatial subset. The FILE_TYPE keyword set to “Classification” only allows selection of a classification band and issues a warning on attempts to select non-classification data. The FILE_TYPE keyword can be used with any of the 20+ types and not just classification images. The BAND_ONLY and NO_DIMS keywords control the additional limitation of only allowing a single band with no spatial subset. The returned file ID and band number are retrieved with the FID and POS keywords, respectively. In addition, if the “Cancel” button is selected (e.g., FID[0] = -1), then
return from the current user procedure. Finally, use the TITLE keyword to set the dialog title bar to Classification Input File.

```fortran
; First get the integer file type value using ENVI_FILE_TYPE
ftype = ENVI_FILE_TYPE('Classification')
; Now prompt the user for the classification band
ENVI_SELECT, fid=fid, pos=pos, /band_only, $ /no_dims, file_type=ftype, $   title='Classification Input File'
if (fid[0] eq -1) then return
```

**Notes**

If fid(0) = -1, the cancel button was selected and the user should take the appropriate action.

**See Also**

ENVI_OPEN_FILE, ENVI_PICKFILE
ENVI_SENSOR_TYPE

Use this function to get a string or integer value of a particular sensor type. Inputting an integer value returns the string sensor type and likewise inputting a string sensor type returns the corresponding integer value. The predefined sensor types are shown below and the user can also add sensor types to the file sensor.txt in the menu directory of the release tree.

Calling Sequence

Result = ENVI_SENSOR_TYPE(Sensor_type)

Arguments

Sensor_type

Sensor_type can be either an integer or a string. If Sensor_type is an integer, the function returns the string-format description of the sensor type specified. If Sensor_type is a string, the function returns the integer code describing the sensor type specified.

The following table shows the available sensor types. String sensor type descriptions are case-sensitive and may contain spaces.

Note

Because additional sensor types are likely to be added in future releases of ENVI, it is strongly recommended that you use string descriptors rather than integer descriptors when referencing sensor types. See the example below for details.

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>Integer</th>
<th>Sensor Type</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAR</td>
<td>0</td>
<td>Landsat MSS</td>
<td>21</td>
</tr>
<tr>
<td>ADEOS</td>
<td>1</td>
<td>Landsat TM</td>
<td>22</td>
</tr>
<tr>
<td>ADRG</td>
<td>2</td>
<td>LEWIS HSI</td>
<td>23</td>
</tr>
<tr>
<td>Air Photo</td>
<td>3</td>
<td>MAS</td>
<td>24</td>
</tr>
<tr>
<td>AIRSAR</td>
<td>4</td>
<td>MASTER</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 9-4: ENVI sensor types
<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>Integer</th>
<th>Sensor Type</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASTER</td>
<td>5</td>
<td>MIVIS</td>
<td>26</td>
</tr>
<tr>
<td>AVHRR</td>
<td>6</td>
<td>MODIS</td>
<td>27</td>
</tr>
<tr>
<td>AVIRIS</td>
<td>7</td>
<td>MOMS-02</td>
<td>28</td>
</tr>
<tr>
<td>CASI</td>
<td>8</td>
<td>RADARSAT</td>
<td>29</td>
</tr>
<tr>
<td>DEM</td>
<td>9</td>
<td>Scanned Image</td>
<td>30</td>
</tr>
<tr>
<td>DMSP</td>
<td>10</td>
<td>SEAWIFS</td>
<td>31</td>
</tr>
<tr>
<td>EARTHWATCH</td>
<td>11</td>
<td>SEBASS</td>
<td>32</td>
</tr>
<tr>
<td>ERS-1</td>
<td>12</td>
<td>SIR-C</td>
<td>33</td>
</tr>
<tr>
<td>ERS-2</td>
<td>13</td>
<td>SPIN-2</td>
<td>34</td>
</tr>
<tr>
<td>GER63</td>
<td>14</td>
<td>SPOT</td>
<td>35</td>
</tr>
<tr>
<td>GEOSCAN</td>
<td>15</td>
<td>TIMS</td>
<td>36</td>
</tr>
<tr>
<td>HYDICE</td>
<td>16</td>
<td>TMS</td>
<td>37</td>
</tr>
<tr>
<td>IRS-1</td>
<td>17</td>
<td>TRWIS III</td>
<td>38</td>
</tr>
<tr>
<td>IRS-2</td>
<td>18</td>
<td>X-SAR</td>
<td>39</td>
</tr>
<tr>
<td>IRS-3</td>
<td>19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 9-4: ENVI sensor types*

**Keywords**

None

**Example**

The following example prints the sensor type string for the file associated with the file ID `FID`. The first command stores the integer file type descriptor in the variable `SENSOR_TYPE`:

```plaintext
envi_file_query, fid, sensor_type=sensor_type
```
The next line stores the string sensor type descriptor in the variable `type_string`:

```python
    type_string = envi_sensor_type(sensor_type)
    print, type_string
```

The following example enters the array DATA into ENVI and sets the sensor type to SPOT. The first line gets the integer sensor type for SPOT. The second line enters the data and sets the SENSOR_TYPE corresponding to SPOT.

```python
    sensor_type = envi_sensor_type('SPOT')
    envi_enter_data, DATA, sensor_type=sensor_type
```

See Also

`ENVI_FILE_QUERY, ENVI_FILE_TYPE`
ENVI_SET_INHERITANCE

Use this routine to return the ENVI inheritance structure. ENVI inheritance allows a newly created ENVI file to automatically inherit properties from the original data. The result of this function is used to set the INHERIT keyword to the ENVI_SETUP_HEAD or ENVI_ENTER_DATA routines. Inheritance has three options, FULL, SPATIAL or SPECTRAL. The FULL keyword inherits all spatial, spectral and other information from the original file. In addition, the FULL keyword can be used with the NO_SPATIAL or NO_SPECTRAL keywords to inhibit inheritance of spatial or spectral information. The SPATIAL keyword inherits only the spatial information from the original file and the SPECTRAL keyword inherits only the spectral information from the original file. See the keyword definitions for more information.

This routine should be used instead of directly accessing the inherit structure.

Calling Sequence

Result = ENVI_SET_INHERITANCE (Fid, Dims [,Pos])

Arguments

Fid

Use this argument to specify the file ID to inherit information from. The value of FID should be the file ID of the original data used to generate the new output image. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure or the FID keyword to ENVI_SELECT. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

Dims

Use this argument to specify the spatial dimensions to inherit information from. The value of Dims should be the values used to compute the new output bands. Dims is a five-element array of long integers with the following definitions:

- Dims[0]: Unused for this function, set to -1.
- Dims[1]: The starting X pixel. (The first pixel is number zero.)
- Dims[2]: The ending X pixel.
- Dims[3]: The starting Y pixel. (The first pixel is number zero.)
- Dims[4]: The ending Y pixel.
**Pos (optional)**

Use this argument to specify an array of band positions, indicating the band numbers to inherit information from. The bands specified by POS should be the bands used to generate the new output image. POS is an array of long integers, ranging from zero to the number of bands-1. The default is to set Pos=[0], the first band.

**Keywords**

**FULL (optional)**

Set this optional keyword to enable full inheritance. The FULL keyword will inherit wavelengths, full width half-max, bad bands list, sensor type, file type, map information, pixel size, XY starting pixel, classification information, spectral library information, Z-plot range, geographic points, default stretch, and default RGB bands. If the FULL keyword is set then the SPATIAL or SPECTRAL keywords are not needed.

**NO_SPATIAL (optional)**

Set this optional keyword to inhibit spatial inheritance. The NO_SPATIAL keyword will turn off inheritance of map information, pixel size, XY starting pixel, and geographic points. The NO_SPATIAL keyword should only be used in conjunction with the FULL keyword.

**NO_SPECTRAL (optional)**

Set this optional keyword to inhibit spectral inheritance. The NO_SPECTRAL keyword will turn off inheritance of wavelengths, full width half-max, bad bands list, and default RGB. The NO_SPECTRAL keyword should only be used in conjunction with the FULL keyword.

**SPATIAL (optional)**

Set this optional keyword to enable spatial inheritance. The SPATIAL keyword will inherit map information, pixel size, XY starting pixel, and geographic points.

**SPECTRAL (optional)**

Set this optional keyword to enable spectral inheritance. The SPECTRAL keyword will inherit wavelengths, full width half-max, bad bands list, and default RGB bands.
Example

The following example will create an inherit structure able to inherit all the information from the original file. This example assumes that FID, DIMS and POS are previously set.

```python
inherit = envi_set_inheritance(fid, dims, pos, /full)
```

The following example will create an inherit structure able to inherit only the spectral information from the original file. This example assumes that FID, DIMS and POS are previously set.

```python
inherit = envi_set_inheritance(fid, dims, pos, /spectral)
```

See Also

- ENVI_ENTER_DATA
- ENVI_SETUP_HEAD
ENVI_SETUP_HEAD

Use this routine to create the ENVI header information for a disk file. In order for
disk files to be used in ENVI they need the associated ENVI header information. The
required keywords (DATA_TYPE, INTERLEAVE, NB, NL, NS, OFFSET) define
the most primitive set of information needed to access the data in the file. The
optional keywords allow the definition of more specific file information.

In memory, data files do not need to call ENVI_SETUP_HEAD instead they use
ENVI_ENTER_DATA.

Calling Sequence

ENVI_SETUP_HEAD

Keywords

BBL (optional)

Use this optional keyword to specify an array of ones and zeros representing the good
and bad bands. The number of elements in BBL must be equal to the number of bands
in the image. A one represents a good band and a zero represents a bad band.

BNAMES (optional)

Use this optional keyword to specify the band names assigned to the data. BNAMES
is a string array of number-of-bands band names. The default band names are [band
1, band 2, etc.]

BYTE_ORDER (optional)

Use this optional keyword to specify a named variable that will contain a flag
indicating the byte order of the data. Set byte order to 1 for big-endian data (i.e., data
generated on SUN, SGI, PowerMac. etc.) and to 0 for little-endian data i.e., data
generated on PC x86). The default is the byte order of the machine.

CLASS_NAMES

Use this keyword to specify a string array of class names for classification images.
The first element (class 0) is the “Unclassified” class. CLASS_NAMES should only
be specified if the result is a classification image. CLASS_NAMES is a required
keyword if the file type is equal to ‘classification’, otherwise, it is optional.
DATA_TYPE

Use this keyword to specify the IDL data type of the file, using the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

DEF_BANDS (optional)

A 1 or 3 element array containing which bands to load by default upon opening in ENVI...if 1 elements, then ENVI will load a gray scale. if 3 elements, ENVI will load an RGB... The values are 0 based so to load bands 4,3,2 of a 7 band image, set DEF_BANDS=[3,2,1].

DEF_STRETCH (optional)

Use this optional keyword to specify the default stretch information. Set DEF_STRETCH equal to the value returned from ENVI_DEFAULT_STRETCH_CREATE.

DESCRIPT (optional)

Use this optional keyword to specify a text description of the data or of the type of processing performed.

FILE_TYPE (optional)

Use this optional keyword to specify a named variable that will contain the integer file type value. See “ENVI_FILE_TYPE” on page 417 for details on how to determine the integer file type value.

FNAME

Use this keyword to specify the file name (and path) of the disk file. FNAME is a string variable that ENVI will use to open the file for reading.

FUNC_COMPLEX (optional)

Set this optional keyword to indicate the Complex Lookup Function to determine which image to display for complex data. This keyword should be set to one of the following:

0 - Power (log10 of magnitude)
1 - Magnitude (square root of sum of the squares of the real and imaginary)
2 - Real (real portion of number)
3 - Imaginary (imaginary portion)
4 - Phase (arc tangent of imaginary divided by real)

The default image is Power. Only set this keyword if the IDL data type of the image is 6=complex (2 x 32-bit) or 9=double-precision complex (2 x 64-bit)

**FWHM (optional)**

Use this optional keyword to specify an array of full-width-half-maximum responses for each band. The number of elements in this array is equal to the number of bands in the image.

**GEO_POINTS (optional)**

Use this optional keyword to specify a 16-element double array of geographic coordinates for the four corners. The array is comprised of four groups of X and Y pixel locations and the corresponding latitude and longitude values (negative for South latitude and negative for West longitude). The four groups of points represent the four corners: upper left, upper right, lower left, lower right. The array is defined as follows

```
GEO_POINTS[0:3] -> [X, Y, LAT, LON] - upper left
GEO_POINTS[8:11] -> [X, Y, LAT, LON] - lower left
```

**INFO (optional)**

Use this optional keyword to specify user defined information. INFO is used to store information which can be passed to users spatial and spectral readers. INFO is retrieved from ENVI_FILE_QUERY using the keyword H_INFO which is a handle to the data. Use HANDLE_VALUE and the handle H_INFO to retrieve the data INFO.

**INHERIT (optional)**

Use this optional keyword to specify the file inheritance. Set INHERIT equal to the value returned from ENVI_SET_INHERITANCE.

**INTERLEAVE**

Use this keyword to specify the file’s interleave type. Set to zero if the file is in BSQ format, to one if BIL format, and to two if BIP format.
LOOKUP (optional)

Use this optional keyword to specify a long array of class RGB values. The
LOOKUP array contains one RGB triple for each class specified with the
NUM_CLASSES keyword. The dimensions of the array are [3,NUM_CLASSES]
and the RGB triplet is ordered [R,G,B]. LOOKUP must be set when entering a
classification image.

MAP_INFO (optional)

Use this optional keyword to specify map information. Set MAP_INFO equal to the
structure returned from ENVI_MAP_INFO_CREATE.

NB

Use this keyword to specify the number of bands in the file.

NL

Use this keyword to specify a the number of lines in the file.

NS

Use this keyword to specify the number of samples in the file.

NUM_CLASSES (optional)

Use this optional keyword to specify the number of classes for classification files.
Remember to include the “Unclassified” class (class 0) in the number of classes. This
keyword should only be specified for classification files.

OFFSET

Use this keyword to specify the offset (in bytes) to the start of the data in the file.

OPEN (optional)

Set this optional keyword to add the file to the Available Bands List. The default is
not to add the file to the Available Bands List.

PIXEL_SIZE (optional)

Use this optional keyword to specify the pixel size of images that are not
goreferenced. PIXEL_SIZE is a floating point array of two specifying the X and Y
pixel size respectively.
R_FID (optional)

Use this optional keyword to specify a named variable that will hold the file ID for files that are added to the Available Bands List. The file ID is only set when the OPEN keyword is also set.

READ_PROCEDURE (optional)

Use this optional keyword to specify a named variable that will contain a string array of the procedure names for the spatial and spectral readers, respectively.

SENSOR_TYPE (optional)

Use this optional keyword to specify an integer value related to the sensor type. See “ENVI_SENSOR_TYPE” on page 504 for details on how to determine the integer sensor type value.

SPEC_NAMES (optional)

Use this optional keyword to specify a named variable that will contain a string array of spectral library names. The keyword SPEC_NAMES should only be set for a spectral library file.

UNITS (optional)

Use this optional keyword to specify the PIXEL_SIZE units for images that are not georeferenced. UNITS is an integer value returned from ENVI_TRANSLATE_PROJECTION_UNITS. Georeferenced images do not use this value instead they use the pixel size and units contained in the map information structure.

WL (optional)

Use this optional keyword to specify an array of wavelength values. The number of elements in this array is equal to the number of bands.

WRITE (optional)

Set this optional keyword to write an output header to disk. The default is to not write an output header. It is possible (and valid) to add a file to the Available Bands List (OPEN keyword) and not write the header file to disk. You should always set the WRITE keyword for files that you wish to open in a later ENVI session.
**XSTART (optional)**

Use this optional keyword to specify the X starting sample for the first pixel in the file. The default is zero. XSTART in conjunction with YSTART are used to preserve the spatial reference for subsetted files. When processing a file the XSTART of the output file is typically set to the XSTART of the input file plus the value of DIMS[1] (the starting sample).

**YSTART (optional)**

Use this optional keyword to specify the Y starting line for the first pixel in the file. The default is zero. YSTART in conjunction with XSTART are used to preserve the spatial reference for subsetted files. When processing a file the YSTART of the output file is typically set to the YSTART of the input file plus the value of DIMS[3] (the starting line).

**ZPLOT_AVERAGE (optional)**

Use this optional keyword to specify a two-element long array for the X and Y window size (in pixels) for the Z-Profile. The window size must be “1” or greater. The Z-Profile is formed from the average of the profiles within the specified window. The default window size is [1,1].

**ZPLOT_TITLES (optional)**

Use this optional keyword to specify a two element string array for the X and Y plot titles. The default X title is “Band Number” for images with no wavelength information and “Wavelength” for images with wavelength information. The default Y axis title is “Value”.

**ZRANGE (optional)**

Use this optional keyword to specify a two dimensional array for the lower and upper spectral plot range.

**Example**

The following example will generate a 256x256 three band BSQ byte ramp image using the function BINDGEN. This image will be saved to disk using the filename test.img. Once the file is saved to disk ENVI_SETUP_HEAD will be used to write an ENVI header and add the image to the Available Bands List. This is one of the simplest uses of ENVI_SETUP_HEAD.
The following example will create two classes (plus the Unclassified class) from a single band ramp image, save the file to disk, and enter the resulting classification image into ENVI. The Unclassified class will be Black [0,0,0], the first class will be Red [255,0,0] and the second class will be Yellow [255,255,0]. The class names will be Unclassified, Red, and Yellow. The classification image will have the description “Example Classification Image” and the band name “Ramp Classification.”

; Create a 2D ramp and then classify all values from 20 to 100 in the first class (classification data value equal to one) and classify all values from 101 to 220 into the second class (classification data value equal to two);
; data = BINDGEN(256,256)
class = BYTE((data ge 20 and data le 100) + 2B * (data ge 101 and data le 220))
; Save the classification image to disk;
fname = 'test.img'
openw, unit, fname, /get_lun
writeu, unit, class
free_lun, unit
; Create the classification information
ENVI_SETUP_HEAD, fname=fname, $ns=256, nl=256, nb=3, $interleave=0, data_type=1, $offset=0, /write, /open
class_names = ['Unclassified','Red','Yellow']
lookup = [[0,0,0],[255,0,0],[255,255,0]]
bnames = ['Ramp Classification']
descrip = 'Example Classification Image'
file_type = ENVI_FILE_TYPE('ENVI Classification')

ENVI_SETUP_HEAD, fname=fname, $
ns=256, nl=256, nb=1, $
interleave=0, data_type=1, $
offset=0, num_classes=3, $ 
class_names=class_names, $ 
lookup=lookup, file_type=file_type, $ 
bnames=bnames, descrip=descrip, $
/write, /open

The following example will take a ramp image and assign a geographic projection to
the image with the upper left corner at 15 degrees south, 48 degrees west, and an X
and Y pixel size of one arc second. The map projection will be created using
ENVI_MAP_INFO_CREATE and the resulting structure will use the MAP_INFO
keyword when entering the data into ENVI.

data = BINDGEN(256,256)

fname = 'test.img'
openw, unit, fname, /get_lun
writeu, unit, data
free_lun, unit

mc = [.5D,.5, -48,-15]
ps = [1D/3600, 1D/3600]
units = ENVI_TRANSLATE_PROJECTION_UNITS('Degrees')
map_info = ENVI_MAP_INFO_CREATE(/geographic, $ 
mc=mc, ps=ps, units=units) 

Enter the data into ENVI
ENVI_WRITE

ENVI_SETUP_HEAD

See Also

ENVI_ENTER_DATA
ENVI_STATS_DOIT

Use ENVI_STATS_DOIT to calculate statistics of a data file and optionally display the results or write the results to a file. The calculated statistics include: covariance, data maximum, data minimum, eigenvalues, eigenvectors, histograms, mean, and standard deviation.

Calling Sequence

ENVI_DOIT, ‘ENVI_STATS_DOIT’

Keywords

COMP_FLAG

Set this keyword equal to a bit value indicating the computations to perform. The bit values are logically OR’d together to perform the requested calculations. The first bit will calculate the data maximum, minimum, mean, and standard deviation. Additional bits must be set to calculate histograms and to calculate covariance, eigenvalues, and eigenvectors. The definition of the bits used in COMP_FLAG are as follows:

- bit 0 – enables the calculation maximum, minimum, mean, and standard deviation.
- bit 1 - enables the calculation of histograms.
- bit 2 - enables the calculation of covariance, eigenvalues and eigenvectors.
- bit 3 to 15 - not used.

If only maximum, minimum, mean, and standard deviation are desired then COMP_FLAG should be set to one, otherwise set COMP_FLAG based on the need to compute histograms and covariance, eigenvalues, and eigenvectors. For example, to calculate all the available statistics (maximum, minimum, mean, standard deviation, histograms, covariance, eigenvalues, and eigenvectors) set COMP_FLAG=7.

COV (optional)

Use this optional keyword to specify a named variable that will contain the returned covariance matrix. You must set bit 2 in COMP_FLAG (i.e. COMP_FLAG=4) to generate the covariance matrix.
DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): A pointer to the region of interest. Set to -1 for non ROI items.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

DMAX (optional)

Use this optional keyword to specify a named variable that will contain the array of data maximums, one for each band position.

DMIN (optional)

Use this optional keyword to specify a named variable that will contain the array of data minimums, one for each band position.

EVAL (optional)

Use this optional keyword to specify a named variable that will contain the eigenvalues. You must set bit 2 in COMP_FLAG (i.e. COMP_FLAG=4) to generate the eigenvalues.

EVEC (optional)

Use this optional keyword to specify a named variable that will contain the eigenvector. You must set bit 2 in COMP_FLAG (i.e. COMP_FLAG=4) to generate the eigenvectors.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.
HIST (optional)

Use this optional keyword to specify a named variable that will contain the histogram array. You must set bit 1 in COMP_FLAG (i.e. COMP_FLAG=2) to generate the histogram output.

M_FID

Use this keyword to specify the file ID for the mask file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

M_POS

Use this keyword to specify the band position of the mask band. M_POS is a single long value greater than or equal to zero.

MEAN (optional)

Use this optional keyword to specify a named variable that will contain the array of data means, one for each band position.

PLOT_FLAG (optional)

Set this keyword equal to the value indicated by the type of output desired.

- bit 0 - plot mean, min, max, and +/- standard deviation
- bit 1 - plot standard deviation
- bit 2 - plot the eigenvalues
- bit 3 - plot the histograms
- bit 4 - save the covariance values to an image
- bit 5 to 15 - not used.

PLOTS_PER_WINDOW (optional)

Use this keyword to specify the number of histogram plots plotted in a single plot widget. The default is to use the lesser of the number of bands or 16.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.
**PREC (optional)**

Use this keyword to specify the format for numbers printed in statistics reports. PREC is a two-element array of long integers with the following definitions:

- **PREC(0):** Set to zero (0) to specify fixed-point notation or set to one to specify scientific (exponential) notation.
- **PREC(1):** Set to the desired number of digits to be printed after the decimal point.

**R_FID (optional)**

Use this keyword to specify a named variable that will hold the file ID for the covariance image. The proper bit must be set in PLOT_FLAG in order to save the covariance image.

**REP_NAME (optional)**

Use this optional keyword to specify the output report filename.

**REPORT_FLAG (optional)**

Set this keyword equal to a bit value indicating the type of output report desired. The bit values are logically OR’d together to generate the requested reports. The definition of the bits used in REPORT_FLAG are as follows:

- bit 0 – generate maximum, minimum, mean, and standard deviation report.
- bit 1 - generate histogram report.
- bit 2 - generate covariance, eigenvalues and eigenvectors report.
- bit 3 to 15 - not used.

Use the keywords TO_SCREEN and REP_NAME to specify the location of the resulting report. When requesting histogram reports and/or covariance, eigenvalues, and eigenvectors reports, you must also set the appropriate COMP_FLAG bit(s). For example, to generate a basic statistics (maximum, minimum, mean, and standard deviation) and histogram report to the screen, set REPORT_FLAG=3, COMP_FLAG=3 and TO_SCREEN=1.

**STA_NAME (optional)**

Use this keyword to specify the filename for the output statistics filename. Output statistics files maybe viewed using the View Statistics File utility (see the online help or the *ENVI User's Guide*).
STDV (optional)

Use this optional keyword to specify a named variable that will contain the array of data standard deviations, one for each band position.

TO_SCREEn (optional)

Set this keyword to print the report on the screen. When this keyword is used in conjunction with a non-zero REPORT_FLAG, a text dialog with the requested report will be displayed.

XFAC (optional)

Use this optional keyword to specify a X skip factor for computing statistics. XFAC is a floating point number less than or equal to one. The default is 1.0. For example, to compute statistics using every tenth pixel set XFAC to 10.

YFAC (optional)

Use this optional keyword to specify a Y skip factor for computing statistics. YFAC is a floating point number greater than or equal to one. The default is 1.0. For example, to compute statistics using every tenth line set YFAC to 10.

Example

```plaintext
pro example_envi_stats_doit
    ; First restore all the base save files.
    ; envi, /restore_base_save_files
    ; Initialize ENVI and send all errors and warnings to the file batch.txt
    ; envi_batch_init, log_file='batch.txt'
    ; Open the input file
    ; envi_open_file, 'can_tmr.img', r_fid=fid
    if (fid eq -1) then begin
        envi_batch_exit
        return
    endif
    ; Get the samples, lines and # bands
    ; for the input file.
```

envi_file_query, fid, ns=ns, nl=nl, nb=nb

; Set the dims and pos to calculate statistics for all data (spatially and spectrally) in the file.
dims = [-1l, 0, ns-1, 0, nl-1]
pos = lindgen(nb)

; Calculate the basic statistics and the histogram for the input data file. Print out the calculated information.
envi_doit, 'envi_stats_doit', fid=fid, pos=pos, $
dims=dims, comp_flag=3, dmin=dmin, dmax=dmax, $
mean=mean, stdv=stdv, hist=hist

; print, 'Minimum ', dmin
print, 'Maximum ', dmax
print, 'Mean ', mean
print, 'Standard Deviation ', stdv

; for i=0(nb-1) do begin
  print, 'Histogram Band ', i+1
  print, hist[*,i]
endfor

; Exit ENVI

See Also

ENVI_GET_STATISTICS, ENVI_WRITE_STATISTICS, CLASS_STATS_DOIT
ENVI_SUM_DATA_DOIT

Use this program to calculate a number of statistical parameters on a set of bands. This routine can be used to sum all the bands into an new output band and/or create a number of other statistics as specified by COMPUTE_FLAG. All operates are done pixel by pixel over the selected number of bands.

Calling Sequence

ENVI_DOIT, ‘ENVI_SUM_DATA_DOIT’

Keywords

COMPUTE_FLAG

Use this keyword to specify which of the output bands to calculate. COMPUTE_FLAG is a seven element array of ones and zeros where a one indicates that the corresponding output band should be calculated. COMPUTE_FLAG has the following definitions:

- COMPUTE_FLAG[0] - Compute the sum of the bands defined by POS
- COMPUTE_FLAG[1] - Compute the sum squared of the bands defined by POS
- COMPUTE_FLAG[2] - Compute the mean of the bands defined by POS
- COMPUTE_FLAG[3] - Compute the standard deviation of the bands defined by POS
- COMPUTE_FLAG[4] - Compute the variance of the bands defined by POS
- COMPUTE_FLAG[5] - Compute the skewness of the bands defined by POS
- COMPUTE_FLAG[6] - Compute the kurtosis of the bands defined by POS
- COMPUTE_FLAG[7] - Compute the mean absolute deviation of the bands defined by POS
DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY (optional)

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

OUT_DT

Use this keyword to specify the output data type as either floating point or double precision. OUT_DT is a long integer using the IDL data type conventions (4=floating-point (32-bits), 5 = double-precision floating point (64-bits)).

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.
**R_FID (optional)**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**Example**

```plaintext
pro example_envi_sum_data_doit
;
; First restore all the base save files.
envi, /restore_base_save_files
;
; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Set the keyword to process all the spatial and all the spectral data.
; Set the keyword COMPUTE_FLAG to compute the sum of the bands, the sum squared of the bands, the mean of the bands, and the standard deviation of the bands.
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb
    dims = [-1l, 0, ns-1, 0, nl-1]
    pos = lindgen(nb)
    out_name = 'testimg'
    compute_flag = [1,1,1,1,0,0,0,0]
    out_dt = 4
;
; Call the processing routine to sum the data together.
;
envi_doit, 'envi_sum_data_doit', $ fid=fid, pos=pos, dims=dims, $ out_name=out_name, out_dt=out_dt, $ compute_flag=compute_flag
```
See Also

ENVI_STATS_DOIT
ENVI_SYNTHETIC_COLOR_DOIT

Use this program to calculate a synthetic color image from a single grayscale band. This program uses a low pass filter, high pass filter and saturation value as inputs into a Hue-Saturation-Value (HSV) transform to create a synthetic color image.

Calling Sequence

ENVI_DOIT, ‘ENVI_SYNTHETIC_COLOR_DOIT’

Arguments

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

H_KSIZE

Use this keyword to specify the high pass kernel size for the filter. The high pass filter is the Value for the HSV transform. H_KSIZE is a long integer greater than one.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
**L_KSIZE**

Use this keyword to specify the low pass kernel size for the filter. The low pass filter is the *Hue* for the HSV transform. **L_KSIZE** is a long integer greater than one.

**OUT_BNAME**

Use this keyword to specify a string array of output band names, if desired.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**POS**

Use this keyword to specify a one elements array of the band position to perform the operations on. **POS** is a single element long array ranging from 0 to the number of bands-1.

**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**SAT_VALUE**

Use this keyword to specify a saturation value for the HSV transform. **SAT_VALUE** is a floating point number from zero to one.

### Example

```plaintext
pro example_envi_synthetic_color_doit
  ; First restore all the base save files.
  ; envi, /restore_base_save_files
  ; Initialize ENVI and send all errors
  ; and warnings to the file batch.txt
  ; envi_batch_init, log_file='batch.txt'
  ; Open the input file
  ; envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
  envi_batch_exit
```
return
endif

; Set the keywords DIMS to
; processes all spatial. Set the
; keyword POS to calculate the
; synthetic color for the first
; band. Set the output band
; names to RGB.
;
envi_file_query, fid, ns=ns, nl=nl
dims = [-1, 0, ns-1, 0, nl-1]
pos = [0]
out_bname = ['R','G','B']
out_name = 'testimg'

; Call the synthetic color processing
; routine. Set the low and high pass
; kernels to 10 pixels and use a
; saturation value of .5 for the HSV
; transform.
;
envi_doit, 'envi_synthetic_color_doit', $
  fid=fid, pos=pos, dims=dims, $
  out_name=out_name, out_bname=out_bname, $
  h_ksize=10, l_ksize=10, sat_value=.5, $
  r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end

See Also

RGB_ITRANS_DOIT, RGB_TRANS_DOIT, MUNSELL_DOIT,
MUNSELL_INV_DOIT
**ENVI_TILE_DONE**

This procedure frees the tile request once tile processing is complete.

**Calling Sequence**

ENVI_TILE_DONE, *Tile_id*

**Arguments**

*Tile_id*

The tile ID as returned from ENVI_INIT_TILE.

**Example**

```plaintext
envi_tile_done, tile_id
```

**Notes**

This procedure must be called once the processing routine has completed the tiling process.

**See also**

ENVI_INIT_TILE, ENVI_GET_TILE
ENVI_TOGGLE_CATCH

Use this routine to toggle the ENVI error catching handling mechanism on and off. When toggled off the catch mechanism will not redirect IDL programming or execution errors. Instead they will be handled according to IDL. This routine is useful when debugging user developed ENVI routine.

Calling Sequence

ENVI_TOGGLE_CATCH
ENVI_TRANSLATE_PROJECTION_NAME

ENVI supports a number of different projection types. This function translates between the projection name string and the projection type integer value and vice-versa. When a projection name string is input, this function returns the integer projection type value. When an integer projection type value is input, this function returns the projection name string.

Calling Sequence

Result = ENVI_TRANSLATE_PROJECTION_NAME(Val)

Arguments

Val

Val can be either an integer or a string. If Val is an integer, the function returns the string-format description of the projection name. If Val is a string, the function returns the integer code describing the projection type.

Example

    type = envi_translate_projection_name('State Plane')
    or
    proj_name = envi_translate_projection_name(3)

See Also

ENVI_MAP_INFO_CREATE
ENVI_TRANSLATE_PROJECTION_UNITS

ENVI supports a number of different projection units. This function translates between the projection units string and the projection units integer value and vice-versa. When a projection units string is input, this function returns the integer projection units value. When an integer projection units value is input, this function returns the projection units string.

Calling Sequence

\[ \text{Result} = \text{ENVI\_TRANSLATE\_PROJECTION\_UNITS}(\text{Val}) \]

Arguments

Val

Val can be either an integer or a string. If Val is an integer, the function returns the string-format description of the projection units. If Val is a string, the function returns the integer code describing the projection units.

Note

Because additional unit types are likely to be added in future releases of ENVI, it is strongly recommended that you use string descriptors rather than integer descriptors when referring projection units. See the example below for details on how to do this.

Example

\[ \text{units} = \text{envi\_translate\_projection\_units('Meters')} \]

See Also

ENVI_MAP_INFO_CREATE
ENVI_USER_DEFINED_ANNOTATION

Use this routine to create an ENVI annotation file or append items to an existing annotation file. If the annotation file does not exist ENVI will create a new annotation file and add the specified item. If the annotation file exists each subsequent call to ENVI_USER_DEFINED_ANNOTATION will add another annotation item to the file. The desired item is specified by the appropriate keyword and any associated parameter, keywords and optional keywords.

Calling Sequence

ENVI_USER_DEFINED_ANNOTATION, Name, X, Y

Arguments

Name
A string variable specifying the annotation filename. If Name does not exist ENVI will create a new annotation file. If Name exists ENVI will append the desired item to the annotation file.

X
The X pixel location for the annotation item.

Y
The Y pixel location for the annotation item.

Keywords

ALIGN (optional)
Use this optional keyword to specify the text alignment for annotation objects supporting this keyword. ALIGN is a floating point number with one of the following values:

- 0 = Left justified
- 1 = Centered
- 2 = Right justified
ARROW (optional)

Set this keyword to specify that the annotation item is an arrow. If ARROW is set the X and Y location parameters are not used but you must specify the keywords XPTS and YPTS to define the two end points of the arrow (in image coordinates). Optional keywords that can be used with ARROW include COLOR, FILL_MODE, FILL_ORIEN, FILL_SPACING, LSTYLE, and THICK.

BGCOLOR (optional)

Use this optional keyword to specify the background color index for annotation objects supporting this keyword. BGCOLOR is an integer color index where: 0=black, 1=white, 2=red, etc. Set BGCOLOR equal to -1 for no background.

CHARSIZE (optional)

Use this optional keyword to specify the character size for annotation objects supporting this keyword. CHARSIZE is a floating point value specifying the character size.

COLOR (optional)

Use this optional keyword to specify the color index for annotation objects supporting this keyword. COLOR is an integer color index into the graphic colors where the default colors are, 0=black, 1=white, 2=red, etc.

COL_RAMP (optional)

Set this keyword to specify that the annotation item is a color ramp. If COL_RAMP is set the X and Y location parameters must be set in image coordinates. Optional keywords that can be used with COL_RAMP include BGCOLOR, COLOR, FONT, CHARSIZE, RAMP_LENGTH, RAMP_INC, RAMP_MAX_VAL, RAMP_MIN_VAL, RAMP_ORIEN, RAMP_PRECISION, RAMP_WIDTH, and THICK.

FILL_MODE (optional)

Use this optional keyword to specify the IDL fill style for annotation objects supporting this keyword. FILL_MODE is an integer specifying the IDL fill style.

FILL_ORIEN (optional)

Use this optional keyword to specify the rotation of fill for annotation objects supporting this keyword. FILL_ORIEN is an integer greater than 1 specifying the fill rotation.
FILL_SPACING (optional)
Use this optional keyword to specify the spacing of fill for annotation objects supporting this keyword. FILL_SPACING is a floating point number indicating the fill spacing. This keyword is only used for FILL_MODE greater than one.

FONT (optional)
Use this optional keyword to specify the font for annotation objects supporting this keyword. FONT is a string value specifying either a hershey font (e.g., “!3” -“!20”) or a valid name for a true type font (e.g., “Courier”).

IMAGE (optional)
Use this keyword to specify and annotation image. IMAGE is a byte array of (NS, NL) for grayscale or (NS, NL, 3) for RGB images. If IMAGE is set the X and Y location parameters must be set in image coordinates and they reference the lower left corner of IMAGE. You must also specify the and the keyword KEY_COLORS and KEY_NAMES must be specified. Optional keywords that can be used with MAP_KEY include BGCOLOR, CHARSIZE, COLOR, FONT, KEY_FILL_MODE, KEY_FILL_ORIEN, KEY_FILL_SPACING, and THICK.

LSTYLE (optional)
Use this optional keyword to specify the IDL line style for annotation objects supporting this keyword. LSTYLE is an integer specifying the IDL line style.

MAP_KEY (optional)
Set this keyword to specify that the annotation item is a map key. If MAP_KEY is set the X and Y location parameters must be set (in image coordinates) and the keyword KEY_COLORS and KEY_NAMES must be specified. Optional keywords for MAP_KEY are BGCOLOR, CHARSIZE, COLOR, FONT, KEY_FILL_MODE, KEY_FILL_ORIEN, KEY_FILL_SPACING, and THICK.

ORIEN (optional)
Use this optional keyword to specify the orientation for annotation objects supporting this keyword. ORIEN is an integer value specifying the clockwise rotation angle.

POLYGON (optional)
Set this keyword to specify that the annotation item is a polygon. If POLYGON is set the X and Y location parameters are not used but you must specify the keywords XPTS and YPTS to define the vertices of the polygon (in image coordinates). Optional keywords that can be used with POLYGON include COLOR,
FILL_MODE, FILL_ORIEN, FILL_SPACING, HEAD_ANGLE, HEAD_PCT, LSTYLE, and THICK.

**POLYLINE (optional)**

Set this keyword to specify that the annotation item is a polyline. If POLYLINE is set the X and Y location parameters are not used but you must specify the keywords XPTS and YPTS to define the polyline vertices (in image coordinates). Optional keywords that can be used with POLYLINE include COLOR, FILL_MODE, FILL_ORIEN, FILL_SPACING, LSTYLE, and THICK.

**SCALE_BARS (optional)**

Set this keyword to specify that the annotation item is a scale bar. If SCALE_BARS is set you must specify the X and Y location parameters (in image coordinates) and the PIXEL_SIZE keyword. Optional keywords that can be used with SCALE_BARS include BGCOLOR, COLOR, FONT, SCALE_FLAG, SCALE_HEIGHT, SCALE_INC, SCALE_LENGTH, SCALE_SUB_INC, SCALE_TITLES, and THICK.

**STR_TEXT (optional)**

Use this keyword to specify the text string for the annotation item TEXT. Multiple lines of text must be delineated with “!C!C”, for example ‘Hello!C!CWorld’ will print Hello and World on separate lines.

**TEXT (optional)**

Set this keyword to specify that the annotation item is a text string. If TEXT is set you must specify the X and Y location parameters and the STR_TEXT keyword. Optional keywords that can be used with TEXT include ALIGN, BGCOLOR, CHARSIZE, COLOR, FONT, ORIEN, and THICK.

**THICK (optional)**

Use this optional keyword to specify the line thickness for annotation objects supporting this keyword. THICK is an integer value specifying the line thickness of hershey font characters.
Keywords for ARROW

HEAD_ANGLE (optional)

Use this keyword to specify the angle for the head of an ARROW annotation object. HEAD_ANGLE is a floating point value between greater than zero specifying the angel in degrees of the arrow’s head. The default is 35.

HEAD_PCT (optional)

Use this keyword to specify the size of the head for an ARROW annotation object. HEAD_PCT is a floating point value between zero and one specifying the percentage of the head size verses the overall length. The default is .15.

Keywords for COLOR_RAMP

RAMP_INC (optional)

Use this keyword to specify the number of color ramp increments. RAMP_INC is an integer value specifying number of increments. The default is 3.

RAMP_LENGTH (optional)

Use this keyword to specify the color ramp length in pixels. RAMP_LENGTH is an integer value specifying the color ramp length in pixels. The default is 200.

RAMP_MAX_VAL (optional)

Use this keyword to specify the value associated with the high end of the color ramp. RAMP_MAX_VAL is a floating point value.

RAMP_MIN_VAL (optional)

Use this keyword to specify the value associated with the low end of the color ramp. RAMP_MIN_VAL is a floating point value.

RAMP_ORIEN (optional)

Use this keyword to specify the color ramp orientation. RAMP_ORIEN is one of the following integer values:

- 0 = Horizontal - low to high
- 1 = Vertical - high to low
- 2 = Horizontal - high to low
- 3 = Vertical - low to high
**RAMP_PRECISION (optional)**

Use this keyword to specify the number of precision digits for color ramp values. RAMP_PRECISION is an integer value. The default is 1.

**RAMP_WIDTH (optional)**

Use this keyword to specify the color ramp width in pixels. RAMP_WIDTH is an integer value specifying the color ramp width in pixels. The default is 25.

**Keywords for MAP_KEY**

**KEY_FILL_MODE (optional)**

Use this keyword to specify the IDL fill mode for map key polygons. KEY_FILL_MODE is an integer value indicating the IDL fill mode. The default is zero, no fill.

**KEY_FILL_ORIEN (optional)**

Use this keyword to specify the orientation for fill. KEY_FILL_ORIEN is an integer value greater than one indicating the rotation for the fill. The default is zero, no rotation.

**KEY_FILL_SPACING (optional)**

Use this keyword to specify the spacing for fill. KEY_FILL_SPACING is floating point value indicating the fill spacing. KEY_FILL_SPACING is only valid when KEY_FILL_MODE is greater than one. The default is .1.

**Keywords for SCALE_BARS**

**SCALE_FLAG (optional)**

Use this keyword to specify which units are active for a scale bar. SCALE_FLAG is a four element bytarr of zeros and ones where a one make the corresponding units active. The elements of SCALE_FLAG correspond to the units as follows [km, miles, meters, feet]. The default is [1,0,0,0] displaying only kilometer units.

**SCALE_HEIGHT (optional)**

Use this keyword to specify the height in pixels for the scale bars. SCALE_HEIGHT is a integer value specifying the scale bar height in pixels. The default is 25.
SCALE_INC (optional)

Use this keyword to specify the number of increments for each type of scale bar. SCALE_INC is a four element integer array specifying the number of increments for each of the units, [km, miles, meters, feet]. The default is \[1,1,1,1\].

SCALE_LENGTH (optional)

Use this keyword to specify the length in map scale for each type of scale bar. SCALE_LENGTH is a four element floating point array specifying the length in map scale for each of the units, [km, miles, meters, feet].

SCALE_SUB_INC (optional)

Use this keyword to specify the number of sub-increments for the first unit of a given scale bar. SCALE_SUB_INC is a four element integer array specifying the number of increments for each of the units, [km, miles, meters, feet]. The default is \[4,4,4,4\].

SCALE_TITLE (optional)

Use this keyword to specify the titles for each scale bar. SCALE_TITLE is a four element string array specifying the titles for each of the units, [km, miles, meters, feet]. The default is \["Kilometers", "Miles", "Meters", "Feet"\].
ENVI_WRITE_DBF_FILE

Use this routine write a DBF attribute file for an ENVI Vector File (EVF). Attributes are used to describe each record in an EVF file and there is a one-to-one correspondence between EVF records and DBF records. The fields in an attribute record are a mix of numbers (byte, integer, long and floating point) and/or strings. All attribute records in a single DBF file have the same field structure (i.e., an array of identical structures).

Calling Sequence

ENVI_EVF_INFO, Filename, Attributes

Arguments

Filename

Use this keyword to specify the output DBF file name.

Attributes

An anonymous structure of vector attributes records where each structure element is an attribute field. There must be one attribute record for each EVF record. The structure tag names are used as the attribute field names (the maximum field name length is 11 characters). There can be any number of attribute fields comprised of numbers (byte, integer, long and floating point) and/or strings.

Example

```
pro example_envi_write_dbf_file
  ; Allocate 10 records for the database file
  ;
  rec = replicate({point_num:0l, point_id:''},10)
  ;
  ; Fill the records with the attributes
  ;
  for i=0,9 do begin
    rec[i].point_num = i + 1
    rec[i].point_id = 'Point ' + string(i+1)
  endfor
  ;
  ; Save the dbf file
  ;
  envi_write_dbf_file, 'test.dbf', rec
```

See Also

ENVI_EVF_DEFINE_INIT, ENVI_EVF_DEFINE_ADD_RECORD,
ENVI_EVF_DEFINE_CLOSE, ENVI_EVF_OPEN,
ENVI_EVF_READ_RECORD, ENVI_EVF_CLOSE
ENVI_WRITE_STATISTICS

This procedure is used to write statistics data to an ENVI statistics (.sta) file. The values written to the statistics file are defined using the keyword arguments.

Calling Sequence

ENVI_WRITE_STATISTICS, Filename

Arguments

Filename

The filename of the ENVI statistics (.sta) file to be written.

Keywords

COV (optional)

Use this keyword to specify the covariance for the image. COV is a (nb,nb) array where nb is defined by CPOS. If COV is set then EVAL and EVEC must also be set.

CPOS (optional)

Use this keyword to specify the array of band indexes used to calculate COV, EVAL and EVEC. The number of elements of CPOS define the nb for COV, EVAL, and EVEC and may be different than the number of bands defined by FID. The default is that COV, EVAL, and EVEC are calculated for all band in the file defined by FID.

DATA_TYPE (optional)

Use this keyword to specify the image data type. DATA_TYPE may be different that the data type in the file defined by FID. DATA_TYPE is used during any subsequent display of statistics by ENVI.

DMAX (optional)

Use this keyword to specify an array that contains the image maximums. DMAX is a (nb) array where nb is defined by FID.

DMIN (optional)

Use this keyword to specify an array that contains the image minimums. DMIN is a (nb) array where nb is defined by FID.
**EVAL (optional)**

Use this keyword to specify the eigen values for the image. EVAL is a (nb,nb) array where nb is defined by CPOS. If EVAL is set then COV and EVEC must also be set.

**EVEC (optional)**

Use this keyword to specify the eigen vectors for the image. EVEC is a (nb) array where nb is defined by CPOS. If EVEC is set then COV and EVAL must also be set.

**FID**

Use this keyword to specify a file ID for defining the overall number of bands in the statistics calculations. COV, EVAL, and EVEC use CPOS to save statistics for fewer than the total number of bands.

**MEAN (optional)**

Use this keyword to specify an array that contains the image mean. MEAN is a (nb) array where nb is equal to the number of bands in the file defined by FID.

**STDV (optional)**

Use this keyword to specify an array that contains the image standard deviation. STDV is a (nb) array where nb is equal to the number of bands in the file defined by FID.

**Example**

This example computes the statistics for a file using ENVI_STATS_DOIT and then output the result to an ENVI statistics (.sta) file.

```plaintext
dims = [-11, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
; Call the doit
envi_stats_doit, fid=fid, pos=pos, dims=dims, $
  dmin=dmin, dmax=dmax, mean=mean, stdv=stdv, $
  comp_flag=0, report_flag=1

; Save the results
envi_write_statistics, 'test.sta', fid=fid, dmin=dmin, dmax=dmax, $
  mean=mean, stdv=stdv
```
See Also

ENVI_STATS_DOIT, ENVI_GET_STATISTICS
FFT_DOIT

Use this program to perform forward Fast Fourier Transform of images.

Calling Sequence

ENVI_DOIT, ‘FFT_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.
POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands - 1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```
pro example_fft_doit
    ; First restore all the base save files.
    ; envi, /restore_base_save_files
    ; Initialize ENVI and send all errors
    ; and warnings to the file batch.txt
    ; envi_batch_init, log_file='batch.txt'
    ; Open the input file
    ; envi_open_file, 'bhtmref.img', r_fid=fid
    if (fid eq -1) then begin
        envi_batch_exit
        return
    endif
    ; Set the keywords. We will perform the
    ; FFT on all samples and bands in
    ; the file.
    ; envi_file_query, fid, ns=ns, nl=nl, nb=nb
    dims = [-1, 0, ns-1, 0, nl-1]
    pos = lindgen(nb)
    out_name = 'testfft'
    ; Perform the FFT
    ; envi_doit, 'fft_doit', $
    ; fid=fid, pos=pos, dims=dims, $
    ; out_name=out_name, r_fid=r_fid
    ; Exit ENVI
```
; env_batch_exit
end
**FFT_INV_DOIT**

Use this program to apply an FFT filter image and perform the inverse FFT.

**Calling Sequence**

```plaintext
ENVI_DOIT, 'FFT_INV_DOIT'
```

**Keywords**

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**FILTER_FID**

Use this keyword to specify the file ID of a filter image generated by the ENVI_FILTER_DOIT program.

**FILTER_POS**

Use this keyword to specify the band number for the filter image.

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_DT
Use this keyword to specify the output file data type. OUT_DT should be an integer value matching one of the standard IDL data types: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Notes
If a “User Defined” filter is selected, then the filter must exist as an ENVI annotation file prior to starting this function.

Example

```
pro example_fft_inv_doit
;
; First restore all the base save files.
;
envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
;
envi_batch_init, log_file='batch.txt'
;
```
; Open the input file

envi_open_file, 'testfft.img', r_fid=fid
envi_open_file, 'testfft.img', r_fid=f_fid
if (fid eq -1 or f_fid eq -1) then begin
    envi_batch_exit
    return
endif

; Set the keywords. We will perform the inverse FFT on all samples and all bands in the file.

envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
f_pos = [0]

; Perform the FFT

envi_doit, 'fft_inv_doit', $
    fid=fid, pos=pos, dims=dims, $
    filter_fid=f_fid, filter_pos=f_pos, $
    out_dt=4, out_name=out_name, $
    r_fid=r_fid

; Exit ENVI

envi_batch_exit
end
GAINOFF_DOIT

Use this program to apply a gain and offset to each input band.

Calling Sequence

ENVI_DOIT, ‘GAINOFF_DOIT’

Keywords

DIMS
Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID
Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

GAIN
Set this keyword to an array of gain values, one for each element in POS.

IN_MEMORY
Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OFFSET
Set this keyword to an array of offset values, one for each element in POS.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_gainoff_doit
;
; First restore all the base save files.
; envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
;
envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
  envi_batch_exit
  return
endif
;
; Set the keywords. We will perform the
; gain and offset correction on all
; samples and all bands in the file.
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb
  dims = [-1, 0, ns-1, 0, nl-1]
  pos = lindgen(nb)
  out_name = 'testimg'
  gain = [2.00, 1.33, 1.20, $ 1.11, 2.60, 3.12]
  offset = [12.33, 1.10, 6.00, $ ]
```
1.55, 5.32, 4.05]

; Perform the empirical line calibration
envi_doit, 'gainoff_doit', $
   fid=fid, pos=pos, dims=dims, $
   gain=gain, offset=offset, $
   out_name=out_name, r_fid=r_fid

; Exit ENVI
envi_batch_exit
end

; Call the doit
gainoff_doit, fid=fid, pos=pos, dims=dims, out_name=out_name, $
   gain=gain, offset=offset, r_fid=r_fid, in_memory=0

; Exit ENVI
envi_batch_exit
free_lun, lunit
end
GEN_IMAGE_DOIT

Use this program to generate test images.

**Calling Sequence**

ENVI_DOIT, ‘GEN_IMAGE_DOIT’

**Keywords**

**DATA_TYPE**

Use this keyword to specify the output data type.

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**MAX_VAL**

Use this keyword to specify the maximum value for methods 1, 2, 3 and 5.

**MEAN**

Use this keyword to specify the mean value for method 4.

**METHOD**

Set this keyword to specify which type of test image to generate. One of the following:

- 0 - Constant
- 1 - Horizontal Ramp
- 2 - Vertical Ramp
- 3 - Random Noise (uniform)
- 4 - Random Noise (normal)
- 5 - Gaussian Point Spread Function

**MIN_Val**

Use this keyword to specify the minimum value for methods 1, 2, 3 and 5.
NB
Use this keyword to specify the number of bands in the output image.

NL
Use this keyword to specify the number of lines in the output image.

NS
Use this keyword to specify the number of samples in the output image.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

SEED (optional)
Use this keyword to specify an optional random number seed for methods 3 and 4.

SIGMA
Use this keyword to specify the gaussian sigma value for methods 4 and 5. For method 5 +/- one sigma is equal to the number of samples.

VAL
Use this keyword to specify the constant value for Constant images.

Example

```plaintext
pro example_gen_image_doit
; First restore all the base save files.
; envi, /restore_base_save_files
; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; Set the keywords.
```
ns = 512L
nl = 512L
nb = 1
method = 3
min_val = 0
max_val = 1024
data_type = 2
out_name = 'testimg'
;
; Generate the test image
;
envi_doit, 'gen_image_doit', $
    ns=ns, nl=nl, nb=nb, method=method, $
    min_val=min_val, max_val=max_val, $
    data_type=data_type, out_name=out_name, $
    r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
HANDLE_VALUE

The HANDLE_VALUE procedure returns or sets the value of an existing handle.

Calling Sequence

HANDLE_VALUE, ID, Value

Arguments

ID

A valid handle ID.

Value

When using HANDLE_VALUE to return an existing handle value (the default), Value is a named variable in which the value is returned.

When using HANDLE_VALUE to set a handle value, Value is the new value. Note that handle values can have any IDL data type and organization.

Keywords

NO_COPY

By default, HANDLE_VALUE works by making a second copy of the source data. Although this technique is fine for small data, it can have a significant memory cost when the data being copied is large.

If the NO_COPY keyword is set, HANDLE_VALUE works differently. Rather than copy the source data, it takes the data away from the source and attaches it directly to the destination. This feature can be used to move data very efficiently. However, it has the side effect of causing the source variable to become undefined. On a retrieve operation, the handle value becomes undefined. On a set operation, the variable passed as Value becomes undefined.

SET

Set this keyword to assign Value as the new handle value. The default is to retrieve the current handle value.
Example

The following commands demonstrate the two different uses of HANDLE_VALUE:

; Retrieve the value of handle1 into the variable current.
HANDLE_VALUE, handle1, current
; Set the value of handle1 to a 2-element integer vector.
HANDLE_VALUE, handle1, [2,3], /SET
HIST_EXPORT_DOIT

Use this routine to output an image using a supplied Lookup Table (LUT). A lookup table is a method of changing input image DN to new output DN. The lookup table is specified by the LUT keyword. Each input pixel is used to calculate the index into the LUT, and the corresponding value in the LUT becomes the new output value. The index into the LUT is calculated from the input data value, the input data minimum (I_MIN), and the input binsize (I_BINSIZE) by the following formula:

\[
\text{Output value} = \text{LUT}\left[\frac{\text{input value} - \text{I_MIN}}{\text{I_BINSIZE}}\right]
\]

Additionally, the output DNs can be adjusted by the desired output minimum and maximum, which are specified by the keywords O_MIN and O_MAX respectively.

Calling Sequence

ENVI_DOIT, ‘HIST_EXPORT_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

LUT

Use this keyword to specify the lookup table array. The number of elements of LUT is determined by the input data range and the input binsize, I_BINSIZE, using the following formula:
# elements lut = (input data maximum – input data minimum) / I_BINSIZE + 1

**I_BINSIZE**

Use this keyword to specify the input binsize. The binsize is the width in DN that a group of input pixels will map to a single output pixel. For integer data, I_BINSIZE must be an integer greater than or equal to one; for floating point data, I_BINSIZE must be greater than zero.

**I_MIN**

Use this keyword to specify the input data minimum.

**IN_MEMORY (optional)**

Set this optional keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk and OUT_NAME must be specified. If IN_MEMORY is set, then OUT_NAME is not used.

**O_MAX**

Use this keyword to specify the desired output data maximum. O_MAX is a single value with the same data type as OUT_DT.

**O_MIN**

Use this keyword to specify the desired output data minimum. O_MIN is a single value with the same data type as OUT_DT.

**OUT_BNAME (optional)**

Use this optional keyword to specify a string array of output band names, if desired.

**OUT_DT**

Use this keyword to specify the output data type, using the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.
POS

Use this keyword to specify a single band position, which indicates the band number on which to perform the operation. POS is a single long integer that ranges from zero to (number of bands)-1.

R_FID (optional)

Use this optional keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data. A valid file ID is a long integer with a value greater than zero and an invalid file ID is specified as “-1”.

Example

This example will calculate a new byte image using a square root lookup table. Since we have byte data and we do not want to change the dynamic range of the input data, we will set the data minimums to zero and the data maximum to 255. We will use a binsize of “1”—allowing each input pixel to map to its square root value. The LUT must have 256 entries and the value of each entry is just the square root of the index.

This example uses the following files found in the DATA directory of the ENVI installation:

- bhtmref.img
- bhtmref.hdr

```
pro example_hist_export_doit
;
; First restore all the base save files.
; envi, /restore_base_save_files
;
; Initialize ENVI and send all errors and warnings to the file batch.txt
;
; envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
; envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Setup the keywords to the
```
; processing routine
;
envi_file_query, fid, ns=ns, nl=nl, $
   data_type=out_dt
out_name = 'testimg'
dims = [-1, 0, ns-1, 0, nl-1]
pos = [0]
;
; Since we are converting byte data
; and do not wish to change the range
; of the input data we will set the
; data mins to 0 and the max to 255.
; An input binsize of one is used
; to map each input pixel to a
; output value.
;
o_min = 0
o_max = 255
i_min = 0
i_binsize = 1
lut = sqrt(lindgen(256))
;
; Call the doit
;
envi_doit,'hist_export_doit', fid=fid, $
   pos=pos, dims=dims, out_name=out_name, $
   out_dt=out_dt, i_min=i_min, o_min=o_min, $
   o_max=o_max, lut=lut, i_binsize=i_binsize, $
   r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
MAGIC_MEM_CHECK

MAGIC_MEM_CHECK is a memory management function to clear out the necessary cache for in-memory functions. It should always be called before the processing function. It returns a structure containing the tags IN_MEMORY and OUT_NAME, which are then passed to the processing function (doit).

If IN_MEMORY is set to zero, OUT_NAME is a string array containing a valid file name. If IN_MEMORY is set to one, the user wishes to perform the operation in memory. If IN_MEMORY is set to minus one, the OUT_NAME string may not be a valid file name.

Calling Sequence

Result = MAGIC_MEM_CHECK()

Keywords

**DIMS**

Use this keyword to specify a five-element array holding the data dimensions. The elements are defined as follows:

- DIMS(0): a pointer to the ROI (set to “-1” if no ROI is selected).
- DIMS(1): the starting X coordinate.
- DIMS(2): the ending X coordinate.
- DIMS(3): the starting Y coordinate.
- DIMS(4): the ending Y coordinate.

**FID**

Use this keyword to specify the file ID of the selected file.

**IN_MEMORY**

Use this keyword to specify the value of memory processing. Typically, IN_MEMORY is the in_memory tag name returned by the compound widget WIDGET_OUTFM. If WIDGET_OUTFM is not used then set IN_MEMORY to one when storing the result in memory and to zero otherwise.
OUT_NAME
Use this keyword to specify the value of the output file name. Typically this is the value returned from the compound widget WIDGET_OUTF or WIDGET_OUTFM.

OUT_DT
Use this keyword to specify the processing output image data type, using the following IDL convention 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

NB
Use this keyword to specify the number of bands in the output image.

NO_WARNING (optional)
Set this optional keyword to inhibit the posting of warning messages if the required output memory exceeds the available ENVI cache.

Example
After getting the processing parameters call MAGIC_MEM_CHECK to make sure that in memory items will not exceed the total cache size. After calling MAGIC_MEM_CHECK use the values stored in M_RES.IN_MEMORY and M_RES.OUT_NAME for the in memory processing flag or the output filename. If M_RES.CANCEL is set then the user has canceled the operation.

```idl
m_res=magic_mem_check(fid=fid, dims=dims, out_dt=1,$
       nb=n_elements(pos), out_name=result.outf.name,$
       in_memory=result.outf.in_memory)
if (m_res.cancel) then return
in_memory = m_res.in_memory
out_name = m_res.out_name
```
MATCH_FILTER_DOIT

Use this program to perform match filtering.

**Calling Sequence**

ENVI_DOIT, ‘MATCH_FILTER_DOIT’

**Keywords**

**COV**

Use this keyword to specify the input data covariance matrix. Typically this is the covariance of the data with the optional mask applied.

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**ENDMEM**

Use this keyword to specify the endmembers array. ENDMEM is a floating point array (nb, # end members).

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
Chapter 9: ENVI Routines

MEAN
Use this keyword to specify the input data mean. Typically this is the mean of the data with the optional mask applied.

M_FID
Use this keyword to specify the file ID for the mask file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

M_POS
Use this keyword to specify the band position of the mask band. M_POS is a single long value greater than or equal to zero.

O_MAX
Use this keyword to specify the output data maximum to use for conversion to byte. O_MAX is only used when the output data type, OUT_DT, is set to 1 (byte).

O_MIN
Use this keyword to specify the output data minimum to use for conversion to byte. O_MIN is only used when the output data type, OUT_DT, is set to 1 (byte).

OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_DT
Use this keyword to specify the output file data type. OUT_DT should be an integer value matching the IDL data type: 1-byte or 4-float.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify a one-dimensional array of band positions indicating the band numbers on which to perform the operations. POS is a long array ranging from 0 to the number of bands-1.
**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**Example**

```plaintext
pro example_match_filter_doit
  ; First restore all the base save files.
  envi, /restore_base_save_files
  ; Initialize ENVI and send all errors and warnings to the file batch.txt
  envi_batch_init, log_file='batch.txt'
  ; Open the input file
  envi_open_file, 'mof94av.bil', r_fid=fid
  if (fid eq -1) then begin
    envi_batch_exit
    return
  endif
  ; Set the DIMS and POS to keywords to processes all spatial and all spectral data. Output the result to disk.
  envi_file_query, fid, ns=ns, nl=nl, $ nb=nb
dims = [-1l, 0, ns-1, 0, nl-1]
  pos = lindgen(nb)
  out_name = 'testimg'
  ; Read in the endmember text file. The first column are the wavelengths and the next 19 columns are the endmembers. We will use the 19 endmembers for MF.
  ; The endmember data must also be transposed in order to send in a (nb, # endmember) array.
  envi_read_cols, 'm94_em.asc', $ endmem, skip=em_names, /read_skip
```
endmem = transpose(endmem[1:*,*])
out_bname = em_names[2:*]

; Calculate the covariance of the ; input data file.
envi_doit, 'envi_stats_doit', $
  fid=fid, pos=pos, dims=dims, $
  mean=mean, cov=cov, comp_flag=5

; Call the match filter processing ; routine.
envi_doit,'match_filter_doit', $
  fid=fid, pos=pos, dims=dims, $
  mean=mean, cov=cov, endmem=endmem, $
  out_dt=4, out_name=out_name, $
  out_bname=out_bname, r_fid=r_fid

; Exit ENVI
envi_batch_exit
end
MATCH_FILTER_MT_DOIT

Use this program to perform mixture tuned match filtering. Compute both the normal matched filter as well as the mixture tuned match filter infeasibility answer for each endmember.

Calling Sequence

ENVI_DOIT, 'MATCH_FILTER_MT_DOIT'

Keywords

COV

Use this keyword to specify the input data covariance matrix. Typically this is the covariance of the data with the optional mask applied.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

ENDMEM

Use this keyword to specify the endmembers array. ENDMEM is a floating point array (nb, # end members).

EVAL

Use this keyword to specify the input data eigen values. Typically this is the eigen values of the data with the optional mask applied.
**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**MEAN**

Use this keyword to specify the input data mean. Typically this is the mean of the data with the optional mask applied.

**M_FID**

Use this keyword to specify the file ID for the mask file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**M_POS**

Use this keyword to specify the band position of the mask band. M_POS is a single long value greater than or equal to zero.

**O_MAX**

Use this keyword to specify the output data maximum to use for conversion to byte. O_MAX is only used when the output data type, OUT_DT, is set to 1 (byte).

**O_MIN**

Use this keyword to specify the output data minimum to use for conversion to byte. O_MIN is only used when the output data type, OUT_DT, is set to 1 (byte).

**OUT_BNAME**

Use this keyword to specify a string array of output band names, if desired.

**OUT_DT**

Use this keyword to specify the output file data type. OUT_DT should be an integer value matching the IDL data type: 1-byte or 4-float.
OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify a one-dimensional array of band positions indicating the band numbers on which to perform the operations. POS is a long array ranging from 0 to the number of bands-1.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```
pro example_match_filter_mt_doit
  ; First restore all the base save files.
  ; envi, /restore_base_save_files
  ; Initialize ENVI and send all errors
  ; and warnings to the file batch.txt
  ; envi_batch_init, log_file='batch.txt'
  ; Open the input file
  ; envi_open_file, 'mof94av.bil', r_fid=fid
  if (fid eq -1) then begin
    envi_batch_exit
    return
  endif
  ; Set the DIMS and POS to keywords
  ; to processes all spatial and all spectral data. Output the result
  ; to disk.
  ; envi_file_query, fid, ns=ns, nl=nl, $
  nb=nb
dims = [-1l, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
```
; Read in the endmember text file.
; The first column are the
; wavelengths and the next 19
; columns are the endmembers. We will
; use the 19 endmembers for MTFM.
; The endmember data must also be
; transposed in order to send in
; a (nb, # endmember) array.
; envi_read_cols, 'm94_em.asc', $
   endmem, skip=em_names, /read_skip
endmem = transpose(endmem[1:*,*])
out_bname = [$
   'MF Score (' + em_names[2:*] + ')', $
   'Infeasibility (' + em_names[2:*] + ')']
;
; Calculate the statistics for the
; input data file.
;
envi_doit, 'envisatsdoot', $
   fid=fid, pos=pos, dims=dims, $
   mean=mean, cov=cov, eval=eval, $
   evec=evec, comp_flag=5
;
; Call the match filter processing
; routine.
;
envi_doit,'match_filter_mt_doot', $
   fid=fid, pos=pos, dims=dims, $
   mean=mean, cov=cov, eval=eval, $
   evec=evec, endmem=endmem, $
   out_dt=4, out_name=out_name, $
   out_bname=out_bname, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
MATH_DOIT

Use this program to perform mathematical operations on image data.

Calling Sequence

ENVI_DOIT, ‘MATH_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

EXP

Use this keyword to specify the math expression to perform. EXP is a string variable.

FID

Use this keyword to specify an array of file IDs, one for each of the bands in EXP. FID is a long integer array.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.
POSS

Use this keyword to specify an array of band positions corresponding to the band location for each of the bands in EXP. POS is a long integer array.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Notes

Some valid ENVI math expressions are:

- `EXP = 'b1+b1'
- `EXP = 'byte ((float(b1)+float(b2))<255)'`
- `EXP = 'byte(((float(b1)+float(b2)+float(b3))/3.0)'`
- `EXP = '(sin(b1)^2+cos(b2))\cdot tan(b3)'`
- `EXP = 'b1 and b2'
- `EXP = 'b1 xor b2'`

Example

```pro
pro example_math_doit

; First restore all the base save files.
; envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'

; Open the input file
; envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
   envi_batch_exit
   return
endif

; Set the keywords. We will perform the band math on all samples in the file.
; envi_file_query, fid, ns=ns, nl=n1
t_fid = [fid, fid]
dims = [-1, 0, ns-1, 0, nl-1]
```
pos = [0,1]
exp = 'b1 + b2'
out_name = 'testimg'
;
; Perform the band math processing
;
envi_doit, 'math_doit', $
   fid=t_fid, pos=pos, dims=dims, $
   exp=exp, out_name=out_name, $
   r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
MNF_DOIT

Use this program to perform a Minimum Noise Fraction (MNF) transform.

Calling Sequence

ENVI_DOIT, ‘MNF_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

NO_PLOT (optional)

Set this optional keyword to disable the resulting Principal Components eigenvalue plot. When this keyword is set the eigenvalues will not be sent to a plot window.

NOISE_EVEC

Use this keyword to specify the noise eigen vectors. NOISE_EVEC is not needed if keyword SHIFT_DIFF is set.
**NOISE_EVAL**

Use this keyword to specify the noise eigen values. NOISE_EVAL is not needed if keyword SHIFT_DIFF is set.

**NOISE_STA_NAME**

Use this keyword to specify the noise statistics file.

**OUT_BNAME**

Use this keyword to specify a string array of output band names, if desired.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**OUT_NB (optional)**

Use this keyword to specify the number of output bands in the MNF image. OUT_NB is a long integer ranging from one to the number of elements of POS. The default is to use the number of bands specified by POS.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands - 1.

**QUERY (optional)**

Set this keyword to specify that the number of output bands be selected interactively from the eigenvalues.

**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**SD_DIMS**

Use this keyword to specify the spatial dimensions for the shift differencing operation. SD_DIMS only needs to be set if the keyword SHIFT_DIFF is set. SD_DIMS is a five-element array of long integers with the following definitions:

- **DIMS(0): A pointer to the region of interest. Set to -1 for non ROI items.**
ENVI Programmer’s Guide

Chapter 9: ENVI Routines

- **DIMS(1):** The starting X pixel. (The first pixel is number zero.)
- **DIMS(2):** The ending X pixel.
- **DIMS(3):** The starting Y pixel. (The first pixel is number zero.)
- **DIMS(4):** The ending Y pixel.

**SHIFT_DIFF**

Set this keyword to specify that noise statistics be computed by shift differencing.

**STA_NAME**

Use this keyword to specify the statistics file name.

**Example**

```plaintext
pro example_mnf_doit

; First restore all the base save files.
; envi, /restore_base_save_files
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; Open the input file
; envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
; Set the keywords DIMS and POS to
; process all spatial and all spectral
; data.
; envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1l, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
; Call the MNF processing routine
; use the shift difference method
; to calculate the noise statistics.
```
; Output the result to a disk file,
; set the statistics file to a null
; file and do not plot the eigenvalues.
;
envi_doit, 'mnf_doit', 
   fid=fid, pos=pos, dims=dims, 
   sd_dims=dims, out_name=out_name, 
   in_memory=0, sta_name='', /no_plot, 
   shift_diff=1, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
MNF_INV_DOIT

Use this program to perform an inverse Minimum Noise Fraction (MNF) transform. To use the MNF as a filter, first perform a forward MNF without reducing the output number of band. Next, perform the inverse MNF and specify only the number of bands to include in the inverse using the POS keyword.

Calling Sequence

ENVI_DOIT, 'MNF_INV_DOIT'

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI.OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_DT

Use this keyword to specify the inverse MNF output data type.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

STA_NAME

Use this keyword to specify the forward MNF statistics filename. The number of bands output from the inverse MNF is based on the number of bands in the STA_NAME statistics file.

See Also

MNF_DOIT
**MORPH_DOIT**

Use this program to perform morphological filtering including dilatation, erosion, opening, closing.

**Calling Sequence**

ENVI_DOIT, 'MORPH_DOIT'

**Keywords**

**CYCLES**

Use this keyword to specify the integer number of cycles to perform the morphological operation.

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**GRAY**

Set this keyword to perform grayscale, rather than binary operations. Nonzero elements of the kernel parameter determine the shape of the structuring element (neighborhood). If values is not present, all elements of the structuring element are 0, yielding the neighborhood minimum operator for erosion and the maximum operator for dilation.
IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

KERNEL

Use this keyword to specify a structuring element for the morphological process. The elements are interpreted as binary values, either zero or nonzero.

METHOD

Set this keyword to one of the following values to indicate the type of filter to apply:

- 0 - erosion
- 1 - dilation
- 2 - opening
- 3 - closing

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

VALUE

An array of the same dimensions as kernel providing the values of the structuring element. The presence of this keyword implies grayscale erosion. Each element in the structuring element is either subtracted (erosion) or added (dilation) to the associated pixels. The minimum (erosion) or maximum (dilation) is performed.
Example

```plaintext
pro example_morph_doit
  ; First restore all the base save files.
  ;
  envi, /restore_base_save_files
  ; Initialize ENVI and send all errors and warnings to the file batch.txt
  ;
  envi_batch_init, log_file='batch.txt'
  ;
  ; Open the input file
  ;
  envi_open_file, 'can_tmr.img', r_fid=fid
  if (fid eq -1) then begin
      envi_batch_exit
      return
  endif
  ;
  ; Set the keywords. We will perform the morphological filtering on all samples and all bands in the file.
  ;
  envi_file_query, fid, ns=ns, nl=nl, nb=nb
  dims = [-1, 0, ns-1, 0, nl-1]
  pos = lindgen(nb)
  kernel = fltarr(5,5) + 1.
  value = kernel
  out_name = 'testimg'
  ; Perform the morphological opening filter operation on the image.
  ;
  envi_doit, 'morph_doit', $
      fid=fid, pos=pos, dims=dims, $ method=2, gray=1, kernel=kernel, $ value=value, cycles=3, $ out_name=out_name, r_fid=r_fid
  ;
  ; Exit ENVI
  ;
  envi_batch_exit
end
```
MOSAIC_DOIT

Use this routine to overlay two or more images that have overlapping areas (typically georeferenced) or to put together a variety of non-overlapping images (typically pixel-based). Individual bands, entire files, and multi-resolution georeferenced images can be mosaicked. The location of each mosaicked input file is specified using the x0 and y0 keywords. The values to x0 and y0 are in pixels and may require the user to convert the output pixel location for georeferenced images.

Calling Sequence

ENVI_DOIT, ‘MOSAIC_DOIT’

Keywords

BACKGROUND

Use this keyword to specify the output image background value. All pixels outside the image boundary will be set to the value specified by BACKGROUND.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a long integer array [5,#input files] with the following definitions:

- DIMS[0,*]: Unused for this function, set to -1.
- DIMS[1,*]: The starting X pixel. (The first pixel is number zero.)
- DIMS[2,*]: The ending X pixel.
- DIMS[3,*]: The starting Y pixel. (The first pixel is number zero.)
- DIMS[4,*]: The ending Y pixel.

FID

Use this keyword to specify a long integer array of file IDs—one for each file to mosaic. The file ID is the value returned by the R_FID keyword to the ENVI_OPEN_FILE procedure or the FID keyword to ENVI_SELECT.

GEOREF (optional)

Set this keyword to indicate that the data is georeferenced. If the data is georeferenced, the MAP_INFO keyword must be defined.
**IN_MEMORY (optional)**

Set this optional keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk and OUT_NAME must be specified. If IN_MEMORY is set then OUT_NAME is not used.

**MAP_INFO (optional)**

Use this optional keyword to specify map information. Set MAP_INFO equal to the structure returned from `ENVI_GET_MAP_INFO` or `ENVI_MAP_INFO_CREATE`.

**OUT_BNAME (optional)**

Use this optional keyword to specify a string array of output band names, if desired.

**OUT_DT**

Use this keyword to specify a named variable that will contain the IDL data type of the file, using the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**PIXEL_SIZE**

Use this keyword to specify a two-element floating point or double precision array containing the X and Y pixel sizes, respectively. For pixel-based mosaics set the X and Y pixel sizes to one: PIXEL_SIZE=[1.,1.]

**POS**

Use this keyword to specify a long-integer array of band positions to be included in the mosaic. The dimensions of the array must be [#output bands, #files]. The value of minus one (-1) in the POS array is used to indicate that no input band for the associated file is included in the corresponding mosaicked output band. The valid values for POS are 0 to #number of bands (for the corresponding file) plus the value of minus one (-1).
**R_FID (optional)**

Use this optional keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**SEE_THROUGH_VAL (optional)**

Use this optional keyword to set the see-through value for each input file. SEE_THROUGH_VAL is an array of values with the same data type as OUT_DT and has [#files] elements. If a pixel in the input file is equal to its corresponding value of SEE_THROUGH_VAL then that pixel is not transferred to the output mosaic, which allows data stacked underneath to remain visible. If any elements of the array USE_SEE_THROUGH are equal to “1” then the SEE_THROUGH_VAL array must be specified.

**USE_SEE_THROUGH**

Use this keyword to specify an integer array of zeros and ones indicating whether the bands from the current file will use the corresponding SEE_THROUGH_VAL value. A “1” indicates that the input file will use SEE_THROUGH_VAL and a “0” indicates that SEE_THROUGH_VAL will not be used. The number of elements in the USE_SEE_THROUGH array is equal to the number of input files specified by FID.

**XSIZE**

Use this keyword to set the X size of the output image (sample direction). For pixel-based mosaics the XSIZE is in pixels and the corresponding pixel size is [1,1]. For georeferenced images XSIZE is in the same units as PIXEL_SIZE. For example, a 1 Km image in the sample direction would use a pixel size of one meter and a XSIZE of 1000 meters.

**x0**

Use this keyword to specify a long-integer array of X starting pixels, one for each file. x0 is in pixel coordinates and may require the conversion of map locations for georeferenced image to output pixel locations.

**YSIZE**

Use this keyword to set the Y size of the output image (line direction). For pixel-based mosaics the YSIZE is in pixels and the corresponding pixel size is [1,1]. For georeferenced images YSIZE is in the same units as PIXEL_SIZE. For example, a 1 Km image in the line direction would use a pixel size of one meter and a YSIZE of 1000 meters.
y0

Use this keyword to specify a long-integer array of Y starting lines—one for each file. y0 is in pixel coordinates and may require the conversion of map locations for georeferenced image to output line locations.

Example

This example mosaics the first three bands from bhtmref.img and the single band bhdemsub.img side by side into the same output image (pixel-based). The single band DEM will only be placed into the first output band while the TM data will be placed into all three output bands. The data are positioned with a 20-pixel border around the outside and between each of the images. The actual border area will vary because the images are not the same size and the DEM image is only in the first output band. We will use the background value of 255 for the border area. Set the output data type to integer since that is the greater of the two data types. The images are mosaicked side by side so no see-through is needed, but for illustration purposes set see-through to zero for each of the input files.

This example uses the following files found in the DATA directory of the ENVI installation:

- bhtmref.img
- bhtmref.hdr
- bhdemsub.img
- bhdemsub.hdr

pro example_mosaic_doit

; First restore all the base save files.
; envi, /restore_base_save_files
; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; Open the input files
; envi_open_file, 'bhtmref.img', r_fid=tm_fid
if (tm_fid eq -1) then begin
  envi_batch_exit
  return
endif
envi_open_file, 'bhdemsub.img', r_fid=dem_fid
if (dem_fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Build the necessary keywords. We will
; create a new 3 band output image from
; a pixel mosaic of the first three band
; of the TM data and the single band DEM.
;
; envi_file_query, tm_fid, ns=tm_ns, nl=tm_nl
; envi_file_query, dem_fid, ns=dem_ns, nl=dem_nl
fid = [tm_fid, dem_fid]
pos = [[0,1,2],[0,-1,-1]]
dims = [[-1,0, tm_ns-1,0, tm_nl-1], $  
[-1,0,dem_ns-1,0,dem_nl-1]]
out_name = 'testimg'
;
; Determine the placement of the output
; bands. The upper left corner of the
; TM data is at [20,20] and the upper
; lefter corner of the DEM data is [40+tm_ns, 20]
;
x0 = [20,40+tm_ns]
y0 = [20,20]
xsize = tm_ns + dem_ns + 60
ysize = (tm_nl > dem_nl) + 40
pixel_size = [1.,1.]
;
; Although it does not matter (since
; we are placing the files next to each
; other) we will use a see through
; value of zero for each file.
;
use_see_through = [1L,1]
see_through_val = [0L,0]
;
; Call the doit. Use a background value of
; 255 and set the output data type to
; integer.
;
envi_doit, 'mosaic_doit', fid=fid, pos=pos, $
dims=dims, out_name=out_name, xsize=xsize, $  
ysize=ysize, x0=x0, y0=y0, georef=0, $  
out_dt=2, pixel_size=pixel_size, $  
background=255, see_through_val=see_through_val, $  
use_see_through=use_see_through
;
; Exit ENVI
;
envi_batch_exit
end
MSSCAL_DOIT

Use this program to calibrate MSS data to radiance or reflectance using pre-launch characteristics.

Calling Sequence

ENVI_DOIT, ‘MSSCAL_DOIT’

Keywords

CAL_TYPE

Use this keyword to specify the calibration type. Set CAL_TYPE equal to zero to indicate Radiance or to one to indicate Reflectance.

DATE

Use this keyword to specify the calibration numbers to use, depending on the satellite and the date the data was collected. Choose one of the following:

Landsat 2
  • 0 - Before July 16, 1975
  • 1 - After July 16, 1975
Landsat 3
  • 0 - Before June 1, 1978
  • 1 - After June 1, 1978
Landsat 4
  • 0 - Before August 26, 1982
  • 1 - Between August 26, 1982 and March 31, 1983
  • 2 - After March 31, 1983
Landsat 5
  • 0 - Before April 6, 1982
  • 1 - Between April 6, 1982 and November 8, 1984
  • 2 - After November 8, 1984

The default value for DATE is zero.
DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

QCAL_DATE

Set this keyword for Landsat 1, Landsat 2, or Landsat 3 MSS data collected before February 1, 1976, and for Landsat 4 MSS data collected before October 22, 1982. Leave QCAL_DATE unset for all other data.
R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

SAT

Set this keyword to one of the following values to indicate the satellite type:

- 1 - Landsat 1
- 2 - Landsat 2
- 3 - Landsat 3
- 4 - Landsat 4
- 5 - Landsat 5

SUN_ANGLE

Use this keyword to specify a floating-point value between 0.0 and 90.0 corresponding to the sun elevation angle. Sun elevation is only used for reflectance calibration.

Example

```plaintext
pro example_msscal_doit
    ; First restore all the base save files.
    ; envi, /restore_base_save_files
    ; Initialize ENVI and send all errors and warnings to the file batch.txt
    ; envi_batch_init, log_file='batch.txt'
    ; Open the input file
    ; envi_open_file, 'bhtmref.img', r_fid=fid
    if (fid eq -1) then begin
        envi_batch_exit
        return
    endif
    ; Set the keywords. We will perform the
    ; MSS calibration on all samples
    ; and all bands in the file.
```
Chapter 9: ENVI Routines

; envi_file_query, fid, ns=ns, nl=nl
dims = [-1, 0, ns-1, 0, nl-1]
pos = [0,1,2,3]
out_name = 'testimg'
;
; Perform the MSS calibration
;
envi_doit, 'msscal_doit',
    fid=fid, pos=pos, dims=dims,
    date=2, sat=4, sun_angle=76.,
    cal_type=1, qcal_date=0,
    out_name=out_name, r_fi
;
; Exit ENVI
;
envi_batch_exit
**MUNSELL_DOIT**

Use this program to calculate the USGS Munsell Coordinates (HSV) from an RGB image.

**Calling Sequence**

ENVI_DOIT, ‘MUNSELL_DOIT’

**Keywords**

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**FID**

Use this keyword to specify a three-element long-integer array of file IDs for the selected bands. The three elements represent the red, green, and blue bands, respectively.

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**POS**

Use this keyword to specify a three-element long-integer array of band positions. Elements of POS are paired with elements of the FID array.
**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**Example**

```plaintext
pro example_munsell_doit

; First restore all the base save files.
envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
envi_batch_init, log_file='batch.txt'

; Open the input file
envi_open_file, 'can_tmr.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif

; Set the keywords. We will perform the munsell transform on all samples and first three bands in the file.
envi_file_query, fid, ns=ns, nl=nl
t_fid=[fid,fid,fid]
dims = [-1, 0, ns-1, 0, nl-1]
pos = [0,1,2]
out_name = 'testmunsell.img'

; Perform the Munsell transform
envi_doit, 'munsell_doit', $ fid=t_fid, pos=pos, dims=dims, $ out_name=out_name, r_fid=r_fid

; Exit ENVI
envi_batch_exit
end
```
MUNSELL_INV_DOIT

Use this program to calculate an RGB image from USGS Munsell coordinates (HSV).

Calling Sequence

ENVI_DOIT, ‘MUNSELL_INV_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify a three-element long-integer array of file IDs for the selected bands. The three elements represent the red, green, and blue bands, respectively.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify a three-element long-integer array of band positions. Elements of POS are paired with elements of the FID array.
**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**Example**

```
pro example_munsell_inv_doit

; First restore all the base save files.
envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
envi_batch_init, log_file='batch.txt'

; Open the input file
envi_open_file, 'testmunsell.img', r_fid=fid
if (fid eq -1) then begin
   envi_batch_exit
   return
endif

; Set the keywords. We will perform the Munsell transform on all samples and the first three bands in the file.
envi_file_query, fid, ns=ns, nl=nl
t_fid=[fid,fid,fid]
dims = [-1, 0, ns-1, 0, nl-1]
pos = [0,1,2]
out_name = 'testimg'

; Perform the Munsell transform
envi_doit, 'munsell_inv_doit', $
   fid=t_fid, pos=pos, dims=dims, $
   out_name=out_name, r_fid=r_fi

; Exit ENVI
envi_batch_exit
```

NDVI_DOIT

Use this program to create a Normalized Difference Vegetation Index (NDVI).

Calling Sequence

ENVI_DOIT, ‘NDVI_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

O_MAX

Use this keyword to specify the output data maximum. This keyword is only used when OUT_DT is set to byte.

O_MIN

Use this keyword to specify the output data minimum. This keyword is only used when OUT_DT is set to byte.
OUT_DT

Use this keyword to specify the output data type, either 1 for byte or 4 for floating point. All other output data types are invalid.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

POS

Use this keyword to specify an array of zero based two band positions. NDVI is designed to work with TM, MSS, AVHRR, SPOT, and AVIRIS DATA. The following band pairs are used for each of the different data types.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Band Numbers (one based, [IR, Red])</th>
</tr>
</thead>
<tbody>
<tr>
<td>TM</td>
<td>Bands = [4, 3]</td>
</tr>
<tr>
<td>MSS</td>
<td>Bands = [7, 5]</td>
</tr>
<tr>
<td>AVHRR</td>
<td>Bands = [2, 1]</td>
</tr>
<tr>
<td>SPOT XS</td>
<td>Bands = [3, 2]</td>
</tr>
<tr>
<td>AVIRIS</td>
<td>Bands = [51, 29]</td>
</tr>
</tbody>
</table>

Table 9-5: Data Types and Band Numbers.

Make sure to subtract one from each of the band combinations listed in the table.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_ndvi_doit
    ; First restore all the base save files.
    ;
    envi, /restore_base_save_files
```

ENVI Programmer’s Guide
; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
; envi_open_file, 'bhtmref.img', r_fid=fid if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Set the keywords. We will perform the NDVI on all samples and all bands in the file.
;
; envi_file_query, fid, ns=ns, nl=nl
dims = [-1, 0, ns-1, 0, nl-1]
pos = [4,3] - 1
out_name = 'testimg'
;
; Perform the NDVI transform
;
envi_doit, 'ndvi_doit', $
    fid=fid, pos=pos, dims=dims, $
    /check, o_min=0, o_max=255, $
    out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
ON_IOERROR

The ON_IOERROR procedure specifies a statement to be jumped to if an I/O error occurs in the current procedure. Normally, when an I/O error occurs, an error message is printed and program execution is stopped. If ON_IOERROR is called and an I/O related error later occurs in the same procedure activation, control is transferred to the designated statement with the error code stored in the system variables !ERR and !ERROR. The text of the error message is contained in the system variable !ERR_STRING.

The effect of ON_IOERROR can be canceled by using the label “NULL” in the call.

Calling Sequence

ON_IOERROR, Label

Arguments

Label

The label name the program jumps to when an I/O error is encountered.
**PC_ROTATE**

Use this program to calculate the Principal Components Rotation transform.

**Calling Sequence**

ENVI_DOIT, ‘PC_ROTATE’

**Keywords**

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**EVEC**

Use this keyword to specify the eigen vectors.

**EVAL**

Use this keyword to specify the noise eigen values.
FORWARD
Set this keyword to specify that PC_ROTATE perform a Forward Principal Components Rotation otherwise the Inverse rotation is performed.

MEAN
Use this keyword to specify the mean of each band.

NO_PLOT (optional)
Set this optional keyword to disable the resulting Principal Components eigenvalue plot. When this keyword is set the eigenvalues will not be sent to a plot window.

OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_DT
Use this keyword to specify the output data type for the Principle Components Rotation. Valid values of OUT_DT are byte, float or double. Use the IDL convention when specifying OUT_DT (1=byte, 4=float, 5=double).

OUT_NB (optional)
Use this optional keyword to specify the output number of band for the Inverse Principal Components. The value must be less than the number of elements of POS and the default is the number of elements of POS. This value is only used when the keyword FORWARD is not specified or is set to zero.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.
Example

pro example_pc_rotate

; First restore all the base save files.
; envi, /restore_base_save_files
; Initialize ENVI and send all errors and warnings to the file batch.txt.
; envi_batch_init, log_file='batch.txt'
; Open the input file
; envi_open_file, 'mof94av.bil', r_fid=fid if (fid eq -1) then begin
  envi_batch_exit
  return
endif
; Set the DIMS and POS to keywords to processes all spatial and all spectral data. Output the result to disk.
; envi_file_query, fid, ns=ns, nl=nl, $ nb=nb
dims = [-11, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
; Calculate the statistics for the input data file.
; envi_doit, 'environ_stats_doit', $ fid=fid, pos=pos, dims=dims, $ mean=mean, eval=eval, evec=evec, $ comp_flag=5
; Call the Principal Components processing routine.
; envi_doit,'pc_rotate', $ fid=fid, pos=pos, dims=dims, $ mean=mean, eval=eval, evec=evec, $ out_dt=4, out_name=out_name, $ out_nb=nb, r_fid=r_fi
; Exit ENVI
;
envi_batch_exit
end
PPI_DOIT

Use this program to calculate the Pixel Purity Index™ for an image.

Calling Sequence

ENVI_DOIT, ‘PPI_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1s.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

ITERATIONS

Use this keyword to specify the total number of iterations for run. If using a previous PPI_DOIT output then ITERATIONS specifies the additional number of iterations to run.

OUT_BNAME

Use this keyword to specify a string array for the output band name, if desired.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

P_FID

Use this keyword to specify the file ID for a previous output file from the PPI_DOIT. If not using a previous PPI_DOIT file then P_FID must be set to -1.

POS

Use this keyword to specify a three-element long-integer array of band positions. Elements of POS are paired with elements of the FID array.

PREV_ITER

Use this keyword to specify the previous number of iterations performed on the data. PREV_ITER must be set to zero if the data is not restarting a previous processing cycle.

PREV_DATA

Use this keyword to specify an array of the previous Total Pixel Counts. This data is stored in a ".cnt" file from the previous run.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

THRESH

Use this keyword to specify the threshold factor.

Example

```plaintext
pro example_ppi_doit

; First restore all the base save files.
; envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
```

; Open the input file
;
envi_open_file, 'mof94av.bil', r_fid=fid
if (fid eq -1) then begin
   envi_batch_exit
   return
endif
;
; Set the DIMS and POS to keywords
; to processes all spatial and all
; spectral data. Output the result
; to disk.
;
envi_file_query, fid, ns=ns, nl=nl, $  
   nb=nb
   dims = [-1l, 0, ns-1, 0, nl-1]
   pos = lindgen(nb)
   out_name = 'testimg'
;
; Call the PPI processing
;
envi_doit,'ppi_doit', $  
   fid=fid, pos=pos, dims=dims, $  
   p_fid=-1, iterations=10000, $  
   thresh=3.0, prev_iter=0, $  
   out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
RADAR_INC_ANGLE_DOIT

Use this program to calculate an incidence angle for a radar image. No DEM data is used in the calculation of the incidence angle.

Calling Sequence

ENVI_DOIT, ‘RADAR_INC_ANGLE_DOIT’

Keywords

DEGREES (optional)

Set this keyword to specify that the input and output angles are in degrees. Otherwise the angles are assumed to be in radians.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FR_ANGLE

Use this keyword to specify the far range angle for the locations specified by the DIMS array.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

NR_ANGLE

Use this keyword to specify the near range angle for the locations specified by the DIMS array.
LEFT_LOOK

Set this keyword to use a left looking instrument to calculate the incidence angle image.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

RANGE_DIR

Use this keyword to specify the range direction. RANGE_DIR is an integer value with the following definitions:

- 0 - Range by samples
- 1 - Range by lines

Example

```plaintext
pro example_radar_inc_angle_doit
    ; First restore all the base save files.
    ; envi, /restore_base_save_files
    ; Initialize ENVI and send all errors and warnings to the file batch.txt
    ; envi_batch_init, log_file='batch.txt'
    ; Set the keywords.
    ; ns = 512L
    n1 = 512L
    dims = [-1, 0, ns-1, 0, nl-1]
    nr_angle = 20.5
    fr_angle = 28.0
    range_dir = 0
    out_name = 'testimg'
    ; Generate the radar incidence angle image.
```
envi_doit, 'radar_inc_angle_doit', $
   \text{dms=\text{dms}, nr\_angle=nr\_angle,}$
   \text{fr\_angle=fr\_angle, range\_dir=range\_dir,}$
   \text{out\_name=out\_name, r\_fid=r\_fid}$

; Exit ENVI

; envi_batch_exit
end
RATIO_DOIT

Use this program to calculate ratios of selected image bands.

Calling Sequence

ENVI_DOIT, ‘RATIO_DOIT’

Keywords

CHECK

Set this keyword to check for divide by zero errors. Any divide by zero errors are set to zero.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify a one-dimension array of file IDs for the ratio pairs. Each pair is sequential in the list, i.e. (0,1) (2,3) (4,5) etc. FID is a long integer array.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

O_MAX

Use this keyword to specify the output data maximum. This keyword is only used when OUT_DT is set to byte.
O_MIN
Use this keyword to specify the output data minimum. This keyword is only used when OUT_DT is set to byte.

OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_DT
Use this keyword to specify the output data type, either 1 for byte or 4 for floating point. All other output data types are invalid.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify a one-dimension array of band positions for the ratios. POS is an array of long integers, ranging from zero to two times the number of pairs.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```
pro example_ratio_doit

; First restore all the base save files.
envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
envi_batch_init, log_file='batch.txt'

; Open the input file
envi_open_file, 'can_tmr.img', r_fid=fid
if (fid eq -1) then begin
  envi_batch_exit
  return
```
endif
;
; Set the DIMS keywords to processes all spatial data. Use the POS keyword to ratio bands 1 and 4. Output the result to disk.
;
envi_file_query, fid, ns=ns, nl=nl
t_fid=[fid,fid]
dims = [-1, 0, ns-1, 0, nl-1]
pos = [1,4]
out_name = 'testimg'
;
; Perform the ratio calculation
;
envi_doit, 'ratio_doit', $ fid=t_fid, pos=pos, dims=dims, $ out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
REG_WARP_DOIT

Use this program to warp image data. This routine can be used to perform image-to-image or image-to-map registration. The keyword PTS is used to specify the base points and the corresponding input image warp points. The goal is to warp the image so that the warp points align with the base points. If the base and warp points are in pixel coordinates image-to-image registration is performed. If the base points are in map coordinates and the warp points are in pixel coordinates then image-to-map registration is performed. You must also set the TO_IMAGE keyword when performing image-to-image registration.

The keyword METHOD is used to specify the warp method and the desired resampling technique. The keyword DEGREE is used to specify the degree of the warping polynomial for the Polynomial method.

Calling Sequence
ENVI_DOIT, 'REG_WARP_DOIT'

Keywords

BACKGROUND
Use this keyword to specify the output image background value. All pixels outside the warped image boundary will be set to the value specified by BACKGROUND.

DEGREE
Use this keyword to specify the degree of the warping polynomial for the polynomial method. The degree of the polynomial is limited by the number of GCPs and you must make sure that \( \#\text{GCPs} > (\text{degree} + 1)^2 \). For the RST methods set DEGREE equal to one and for the triangulation methods set DEGREE equal to zero.

DIMS
Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
• DIMS(3): The starting Y pixel. (The first pixel is number zero.)
• DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY (optional)

Use this optional keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk and OUT_NAME must be specified.

MAP_INFO (optional)

Use this optional keyword to specify map information. Set MAP_INFO equal to the structure returned from ENVI_MAP_INFO_CREATE.

METHOD

Warping method to use. One of the following:
• 0 - RST with nearest neighbor
• 1 - RST with bilinear
• 2 - RST with cubic convolution
• 3 - Polynomial with nearest neighbor
• 4 - Polynomial with bilinear
• 5 - Polynomial with cubic convolution
• 6 - Triangulation with nearest neighbor
• 7 - Triangulation with bilinear
• 8 - Triangulation with cubic convolution

OUT_BNAME (optional)

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.
PIXEL_SIZE
Use this keyword to specify a two-element array of the X and Y pixel size.

POS
Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

PTS
Use this keyword to specify an array of the X and Y positions of the warp and base tie points. The dimensions of the array are (4, #pairs) where the warp and base points are defined as:
- pts(0,*) - x base points
- pts(1,*) - y base points
- pts(2,*) - x warp points
- pts(3,*) - y warp points

R_FID (optional)
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

TO_IMAGE
Set this keyword to indicate image-to-image registration. This keyword is unset for image-to-map registration.

x0
Use this keyword to specify the warp points X offset. x0 is in pixel coordinates for image to image and map coordinates for image to map registration.

XSIZE
Set this keyword to the output image X size in pixels.

XSTART
Use this keyword to specify the output image X offset.
y0

Use this keyword to specify the warp points Y offset. y0 is in pixel coordinates for image to image and map coordinates for image to map registration.

YSIZE

Set this keyword to the output image Y size in pixels.

YSTART

Use this keyword to specify the output image Y offset.

ZERO_EDGE (optional)

Set this keyword to specify that the edges outside of any triangles be set to the value specified by BACKGROUND. The keyword is only used for Triangulation, METHOD 6, 7, or 8.

Example

This example performs an image to map warp.

```
forward_function envi_map_info_create
pro example_reg_warp_doit
;
; First restore all the base save files.
; envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
;
envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
envi_open_file, 'can_tmr.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Set the keywords
;
out_name='testing'
envi_file_query, fid, ns=ns, nl=n1, nb=nb
```
```plaintext
dims=[-1, 0, ns-1, 0, nl-1]
pos=lindgen(nb)

; Enter the GCPs for the image to map warp.
pts=[
  478104.0, 4427620.0, 456.0, 610.0,
  477904.0, 4441270.0, 366.0, 139.0,
  474024.0, 4434890.0, 271.0, 384.0,
  468094.0, 4427690.0, 107.0, 668.0,
  468524.0, 4441140.0, 37.0, 201.0]

; Reform the GCP array into (4,#pairs).
pts=reform(pts, 4, 5, /overwrite)

; Set up the projection and map information.
ps = [30.0, 30.0]
map_info = envi_map_info_create(/utm,
  mc=[456., 610., 478104.0, 4427620.0],
  zone=13, ps=ps)

; Set x0 and y0 to the upper left pixel
; in output map coords. Set the X and Y
; output map size.

x0 = 466000
y0 = 4447000
xsize=20000.
ysize=15000.

; Performs the warping

envi_doit, 'reg_warp_doit',
  fid=fid, pos=pos, dims=dims,
  map_info=map_info, out_name=out_name,
  method=3, degree=1, pts=pts,
  background=0, to_image=0, xsize=xsize,
  ysize=ysize, xstart=0, ystart=0, x0=x0,
  y0=y0, pixel_size=ps

; Exit ENVI

envi_batch_exit
end
```
RESIZE_DOIT

Use this program to spatially resize image data.

Calling Sequence

ENVI_DOIT, ‘RESIZE_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

INTERP

Use this keyword to specify an integer value corresponding to the interpolation type. Choose one of the following.

- 0 - Nearest Neighbor
- 1 - Bilinear
- 2 - Cubic Convolution
• 3 - Pixel Aggregate

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**OUT_BNAME**

Use this keyword to specify a string array of output band names, if desired.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**RFACT**

Use this keyword to specify a two-element array holding the rebin factors for X and Y. The values of RFACT reflect the IDL convention for resizing data. A value of one does not change the size of the data. Values less than one cause the size to increase, values greater than one cause the size to decrease.

### Example

```idl
pro example_resize_doit
  ; First restore all the base save files.
  ; envi, /restore_base_save_files
  ; Initialize ENVI and send all errors and warnings to the file batch.txt
  ; envi_batch_init, log_file='batch.txt'
  ; Open the input file
  ; envi_open_file, 'can_tmr.img', r_fid=fid
  if (fid eq -1) then begin
    envi_batch_exit
    return
```
endif

; Set the DIMS and POS to keywords ; to processes all spatial and all ; spectral data. Output the result ; to disk.
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
;
; Perform the resize calculation.
; Make the output image twice as ; large in both X and Y. Use ; bilinear interpolation.
;
envi_doit, 'resize_doit', $
   fid=fid, pos=pos, dims=dims, $
   interp=1, rfact=[.5,.5], $
   out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
RGB_GET_BANDS

This procedure brings up an ENVI Band Selection dialog that allows the user to select three bands from the Available Bands List.

### Calling Sequence

```plaintext
RGB_GET_BANDS
```

### Keywords

**DIMS**

Use this keyword to specify a named variable that will contain a five-element array holding the data dimensions. The elements are defined as follows:

- **DIMS(0)**: a pointer to the ROI (set to -1 if no ROI is selected).
- **DIMS(1)**: the starting X coordinate.
- **DIMS(2)**: the ending X coordinate.

---

*[Image of Band Selection dialog]*

**Figure 9-3: A Band Selection dialog.**
• DIMS(3): the starting Y coordinate.
• DIMS(4): the ending Y coordinate.

**FID**

Use this keyword to specify a named variable that will contain the file IDs of the selected files. FID will be an long array of three.

**NO_DIMS (optional)**

Set this optional keyword to disable spatial subsetting on the selected RGB bands.

**POS**

Use this keyword to specify a named variable that will contain a three-element array of the selected band positions.

**STR (optional)**

Use this optional keyword to specify a string array containing the labels for the text boxes used to enter the three band names. The default value for the array is ['R', 'G', 'B'].

**TITLE**

Use this keyword to specify the string used in the title bar of the Band Selection dialog window. Enter the title enclosed in single quotes.

**Example**

This examples uses an RGB selection widget titled *USGS Munsell RGB to HSI Input File* to select three bands. The resulting selection is returned in FID, POS and DIMS.

```plaintext
tstr = 'USGS Munsell RGB to HSI Input File'
rgb_get_bands, title=tstr, fid=fid, pos=pos, dims=dims
if (fid(0) eq -1) then return
```

**Notes**

If fid(0) = -1, the Cancel button was selected and the user should take the appropriate action. There is no requirement that the three bands come from the same file, so the user should make sure to properly handle the three FIDs and band positions.

**See Also**

ENVI_SELECT
RGB_ITRANS_DOIT

Use this program to perform HSV to RGB and HLS to RGB color transforms using IDL functions.

Calling Sequence

ENVI_DOIT, 'RGB_ITRANS_DOIT'

Keywords

DIMS
Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID
Use this keyword to specify a three-element long-integer array of file IDs for the selected bands. The three elements represent the red, green, and blue bands, respectively.

HSV
Set this keyword equal to zero to transform from HSV to RGB, or equal to one to transform from HLS to RGB.

IN_MEMORY
Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify a three-element long-integer array of band positions. Elements of POS are paired with elements of the FID array.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

pro example_rgb_itrans_doit
 ;
 ; First restore all the base save files.
 ;
 envi, /restore_base_save_files
 ;
 ; Initialize ENVI and send all errors
 ; and warnings to the file batch.txt
 ;
 envi_batch_init, log_file='batch.txt'
 ;
 ; Open the input file
 ;
 envi_open_file, 'testhsv.img', r_fid=fid
 if (fid eq -1) then begin
  envi_batch_exit
  return
 endif
 ;
 ; Set the keywords. We will perform the
 ; inverse HSV transform on all samples
 ; and first three bands in the file.
 ;
 envi_file_query, fid, ns=ns, nl=nl
 t_fid=[fid,fid,fid]
 dims = [-1, 0, ns-1, 0, nl-1]
 pos = [0,1,2]
 out_name = 'testimg'
 ;
 ; Perform the inverse HSV transform
 ;
envi_doit, 'rgb_itrans_doit', $
   fid=t_fid, pos=pos, dims=dims, $
   hsv=1, out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
RGB_TRANS_DOIT

Use this program to perform RGB to HSV and RGB to HLS color transforms using IDL functions.

Calling Sequence

`ENVI_DOIT, 'RGB_TRANS_DOIT'`

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify a three-element long-integer array of file IDs for the selected bands. The three elements represent the red, green, and blue bands, respectively.

HSV

Set this keyword equal to zero to transform from RGB to HSV, or equal to one to transform from RGB to HLS. The default is zero.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify a three-element long-integer array of band positions. Elements of POS are paired with elements of the FID array.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_rgb_trans_doit
    ; First restore all the base save files.
    ; envi, /restore_base_save_files
    ; Initialize ENVI and send all errors and warnings to the file batch.txt
    ; envi_batch_init, log_file='batch.txt'
    ; Open the input file
    ; envi_open_file, 'can_tmr.img', r_fid=fid
    if (fid eq -1) then begin
        envi_batch_exit
        return
    endif
    ; Set the keywords. We will perform the RGB HSV transform on all samples and first three bands in the file.
    ; envi_file_query, fid, ns=ns, nl=nl
    t_fid=[fid,fid,fid]
    dims = [-1, 0, ns-1, 0, nl-1]
    pos = [0,1,2]
    out_name = 'testhsv.img'
    ; Perform the HSV transform
    ```
envi_doit, 'rgb_trans_doit', $
    fid=t_fid, pos=pos, dims=dims, $
    hsv=1, out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
**ROC_CURVE_DOIT**

Use this program to compute Receiver Operating Characteristic (ROC) Curves. The ROC Curves compare a series of rule image classification results for different threshold values with ground truth information. A probability of detection ($P_d$) versus probability of false alarm ($P_{fa}$) curve and a $P_d$ verses threshold curve are reported for each selected class (rule band). ENVI can calculate a ROC curve using either a ground truth image or using ground truth regions of interest (ROIs).

**Calling Sequence**

ENVI_DOIT, "ROC_CURVE_DOIT"

**Keywords**

**PLOT_THRESH**

Set this keyword to enable the plotting of the probability of detection (PD) versus threshold plot. The default is to only plot the ROC curves.

**CDIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. CDIMS, CFID, and CPOS define the classification rule image used to compute the ROC curves. CDIMS is a five-element array of long integers with the following definitions:

- CDIMS(0): Unused for this function, set to -1.
- CDIMS(1): The starting X pixel. (The first pixel is number zero.)
- CDIMS(2): The ending X pixel.
- CDIMS(3): The starting Y pixel. (The first pixel is number zero.)
- CDIMS(4): The ending Y pixel.

**CFID**

Use this keyword to specify the file ID for the open classification rule image file. CDIMS, CFID, and CPOS define the classification rule image used to compute the ROC curves. CFID is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. CFID is a long integer with a value greater than zero. An invalid file ID is specified as -1.
CPOS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. CDIMS, CFID, and CPOS define the classification rule image used to compute the ROC curves. CPOS is an array of long integers, ranging from zero to the number of bands-1.

GDIMS (optional)

Use this optional keyword to specify the spatial dimensions on which to perform the operation. GDIMS, GFID, and GPOS define the ground truth classification image used to compute the ROC curves. GDIMS is a five-element array of long integers with the following definitions:

- GDIMS(0): Unused for this function, set to -1.
- GDIMS(1): The starting X pixel. (The first pixel is number zero.)
- GDIMS(2): The ending X pixel.
- GDIMS(3): The starting Y pixel. (The first pixel is number zero.)
- GDIMS(4): The ending Y pixel.

If the keyword ROI_IDS is not set then GDIMS, GFID, GPOS, and GT_PTR must be set.

GFID (optional)

Use this optional keyword to specify the file ID for the open ground truth classification image file. GDIMS, GFID, and GPOS define the ground truth classification image used to compute the ROC curves. GFID is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. GFID is a long integer with a value greater than zero. An invalid file ID is specified as -1. If the keyword ROI_IDS is not set then GDIMS, GFID, GPOS, and GT_PTR must be set.

GPOS (optional)

Use this optional keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. GDIMS, GFID, and GPOS define the ground truth classification image used to compute the ROC curves. GPOS is an array of long integers, ranging from zero to the number of bands-1. If the keyword ROI_IDS is not set then GFID, GPOS, and GT_PTR must be set.

GT_PTR (optional)

Use this keyword to specify as a link between the ground truth class and the corresponding rule image classes. For example, the ground truth class value (the actual pixel value of the ground truth classification image) GT_PTR[k] corresponds
to the class created from the rule image band CPOS[k]. GT_PTR is a long array with
the same number of elements as CPOS. If the keyword ROI_IDS is not set then GDIMS,
GFID, GPOS, and GT_PTR must be set.

**MAX_THRESH**

Use this keyword to specify the threshold maximum for classifying the rule images.
The points per curve, as specified by PPC, are evenly distributed between
MIN_THRESH and MAX_THRESH.

**METHOD**

Set this keyword equal to one of the following values to specify the method for
classifying the rule images.

- 0 - Minimum
- 1 - Maximum

**MIN_THRESH**

Use this keyword to specify the threshold minimum for classifying the rule images.
The points per curve, as specified by PPC, are evenly distributed between
MIN_THRESH and MAX_THRESH.

**PPC**

Use this keyword to specify the number of points per ROC curve. The points are
evenly distributed between MIN_THRESH and MAX_THRESH. PPC is a single
long value greater than or equal to 2.

**PPW**

Use this keyword to specify the maximum number of plot per plot window. PPW is a
single long value greater than zero.

**ROI_IDS (optional)**

Use this optional keyword to specify the ground truth ROIs. ROI_IDS are the
selected ids of the ground truth ROIs returned from the function
ENVI_GET_ROI_IDS. If ROI_IDS is not set then GDIMS, GFID, and GPOS
must be set.

**See Also**

CLASS_CONFUSION_DOIT, CLASS_DOIT, CLASS_RULE_DOIT
**ROI_THRESH_DOIT**

Use this program to create an ROI that contains all pixels with values above, below, or between threshold pixel values.

**Calling Sequence**

ENVI_DOIT, ‘ROI_THRESH_DOIT’

**Keywords**

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**MAX_THRESH**

Use this keyword to specify the maximum threshold. Values below MAX_THRESH are included in the ROI.

**MIN_THRESH**

Use this keyword to specify the minimum threshold. Values above MIN_THRESH are included in the ROI.
POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

ROI_NAME

Use this keyword to specify an output roi name for the generated region of interest.

ROI_COLOR

Use this keyword to specify the output ROI color. ROI_COLOR is a LONG variable representing the index into the ENVI graphic colors. By default ENVI has 17 graphic colors and the index values would range from 0 to 16.

Example

```plaintext
pro example_roi_thresh_doit
pro example_roi_thresh_doit  
;; First restore all the base save files. 
;; envi, /restore_base_save_files  
;; Initialize ENVI and send all errors  
;; and warnings to the file batch.txt  
;; envi_batch_init, log_file='batch.txt'
;; ; Open the input file  
;; envi_open_file, 'can_tmr.img', r_fi=did  
;; if (fid eq -1) then begin  
;; envi_batch_exit  
;; return  
;; endif  
;; ; Set the DIMS keywords to process all  
;; spatial pixels and use the POS  
;; keyword to select the first band  
;; to use for the ROI.  
;; envi_file_query, fid, ns=ns, nl=nl  
;; dims = [-1, 0, ns-1, 0, nl-1]  
;; pos = [0]  
;; roi_name = 'test ROI'  
;; out_name = 'test.roi'  
;;
```
; Perform the ROI threshold
envi_doit, 'roi_thresh_doit', $
   fid=fid, pos=pos, dims=dims, $
   max_thresh=255, min_thresh=30, $
   /no_query, roi_color=5, $
   roi_name=roi_name, roi_id=roi_id
;
; Save the ROI to a file
envi_save_rois, out_name, roi_id
;
; Exit ENVI
envi_batch_exit
end
ROTATE_DOIT

Use this program to rotate images using standard IDL rotations and transpositions.

Calling Sequence

ENVI_DOIT, ‘ROTATE_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.
POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

ROT_TYPE

Use this keyword to specify the rotation type by setting it equal to one of the following values:

- 0 - 0 degrees
- 1 - 90 degrees
- 2 - 180 degrees
- 3 - 270 degrees

TRANSPOSE

Set this keyword to cause the image to be transposed.

Notes

Rotations and transpose are performed as per the IDL conventions.

Example

```idl
pro example_rotate_doit

; First restore all the base save files.
envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
envi_batch_init, log_file='batch.txt'

; Open the input file
envi_open_file, 'can_tmr.img', r_fid=fid
if (fid eq -1) then begin
```
envi_batch_exit
return
endif

; Set the DIMS and POS to keywords
to processes all spatial and all
spectral data. Output the result
to disk.
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
;
; Rotate the image by 90 degrees.
;
envi_doit, 'rotate_doit', $
   fid=fid, pos=pos, dims=dims, $
   rot_type=1, out_name=out_name, $
   r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
RTV_DOIT

Use this program to perform a raster to vector conversion for a classification image or for a contour level on a grayscale image.

Calling Sequence

ENVI_DOIT, ‘RTV_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to an array of ones and zeros indicating the storage location for each of the output vectors. IN_MEMORY is a long array of ones and zero with the same number of elements as VALUES and OUT_NAME.

L_NAMES

Use this keyword to specify a string array of vector level names. L_NAMES has the same number of elements as the keyword VALUES.
OUT_NAME

Use this keyword to specify a string array of output vector file names. The OUT_NAME array must have a valid output filename for each of the zero values in the array IN_MEMORY. OUT_NAME has the same number of elements as the keywords VALUES and IN_MEMORY.

POS

Use this keyword to specify the band position on which to perform the operation. POS is a single element long integer array, ranging from zero to the number of bands-1.

VALUES

Use this keyword to specify an array of classification class values to contour or grayscale contour levels. A vector file will be generated for each element in the VALUES array.

Example

```
pro example_rtv_doit
  ; First restore all the base save files.
  ; envi, /restore_base_save_files
  ; Initialize ENVI and send all errors
  ; and warnings to the file batch.txt
  ; envi_batch_init, log_file='batch.txt'
  ; Open the data and class files
  ; envi_open_file, 'bhtm_sam.img', r_fid=fid
  if (fid eq -1) then begin
    envi_batch_exit
    return
  endif
  ; Set the keyword to convert
  ; all spatial data for
  ; the first three class (the
  ; Unclassified class is the
  ; zero class) to vectors. Each of
  ; the classes will be output to a
  ; separate vector file.
```
```plaintext
envi_file_query, fid, ns=ns, nl=nl, 
  class_names=class_names
pos = [0]
values = [1, 2, 3]
dims = [-1, 0, ns-1, 0, nl-1]
l_name = class_names[values]
out_names = 'testimg_' + $
  strcompress(string(values),/remove_all) + '.evf'
in_memory = lonarr(n_elements(values))
;
; Call the raster to vector
; processing routine.
;
envi_toggle_catch
  envi_doit, 'rtv_doit', 
    fid=fid, pos=pos, dims=dims, 
    values=values, l_name=l_name, 
    in_memory=in_memory, 
    out_names=out_names
;
; Exit ENVI
;
envi_batch_exit
end
```
SAT_STRETCH_DOIT

Use this program to perform a Saturation Stretch on the data.

Calling Sequence

ENVI_DOIT, ‘SAT_STRETCH_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.
POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_sat_stretch_doit

; First restore all the base save files.
; envi, /restore_base_save_files
; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; Open the input file
; envi_open_file, 'can_tmr.img', r_fid=fid
if (fid eq -1) then begin
   envi_batch_exit
   return
endif
; Set the keywords. We will perform the saturation stretch on the first three bands in the file and all spatial pixels.
; envi_file_query, fid, ns=ns, nl=nl
t_fid = [fid,fid,fid]
dims = [-1, 0, ns-1, 0, nl-1]
pos = [0,1,2]
out_name = 'testimg'
; Perform the decorrelation stretch
; envi_doit, 'sat_stretch_doit', $
   fid=t_fid, pos=pos, dims=dims, $
   out_name=out_name, r_fid=r_fid
```
; Exit ENVI

envi_batch_exit
end
SHARPEN_DOIT

Use this program to perform HSV (Hue, Saturation, Value) and Color Normalized sharpening on an image.

Calling Sequence

ENVI_DOIT, ‘SHARPEN_DOIT’

Keywords

F_DIMS

Use this keyword to specify the spatial dimensions of the high resolution image. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

F_FID

Use this keyword to specify the high resolution data file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

F_POS

Use this keyword to specify the band position of the high resolution sharpening band. F_POS is a single long value greater than or equal to zero.

FID

Use this keyword to specify the file IDs for the three bands to sharpen. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.
IN_MEMORY
Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

INTERP
Use this keyword to specify an integer value corresponding to the interpolation type. Choose one of the following.

- 0 - Nearest Neighbor
- 1 - Bilinear
- 2 - Cubic Convolution

METHOD
Use this keyword to specify the sharpening method. METHOD is a long value set to one of the following.

- 0 - HSV sharpening
- 1 - Color Normalized sharpening

OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify the three band positions to perform the operations on. Each element of POS corresponds to the appropriate file in the FID array. POS is a long array of three with each element ranging from 0 to the number of bands-1.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.
Example

```plaintext
pro example_sharpen_doit

; First restore all the base save files.
; envi, /restore_base_save_files
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; Open the input file high resolution file
; envi_open_file, 'bldr_sp.img', r_fid=f_fid
if (f_fid eq -1) then begin
   envi_batch_exit
   return
endif
; Set the keyword DIMS and POS to
; process all spatial and all spectral
; data.
; envi_file_query, f_fid, ns=f_ns, nl=f_nl
; Set the keywords
f_dims = [-1l, 0, f_ns-1, 0, f_nl-1]
f_pos = [0]
; Open the file for the RGB bands
; envi_open_file, 'bldrtm_m.img', r_fid=fid
if (fid eq -1) then begin
   envi_batch_exit
   return
endif
envi_file_query, fid, bnames=bnames
pos = [2,1,0]
rgb_fid = [fid,fid,fid]
out_name = 'testing'
out_bname = ['Red','Green','Blue']
; Call the sharpening processing
; routine.
; envi_doit, 'sharpen_doit',
   fid=rgb_fid, pos=pos, f_fid=f_fid,
```

SHARPEN_DOIT

f_dims=f_dims, f_pos=f_pos, $
out_name=out_name, method=1, $
interp=0, out_bname=out_bname
;
; Exit ENVI
;
envi_batch_exit
end
SIRC_HEADER_DOIT

Use this function to read the SIRC header information used by other SIRC routines. The function returns a 1 if the specified file is a SIRC file otherwise 0 is returned. The desired SIRC information is returned using optional keywords.

Calling Sequence

ENVI_DOIT, ‘SIRC_HEADER_DOIT’

Keywords

AZ_KM (optional)
Use this optional keyword to specify a named variable that will contain the azimuth or Y image size in kilometers.

BAND (optional)
Use this optional keyword to specify a named variable that will contain the band number for the file specified by FNAME. BAND is set to 0 for C-Band and 1 for L-Band.

CANCEL (optional)
Use this optional keyword to specify a named variable that will be set to 0 if the user cancels the read request.

FILENAME
Set this keyword to specify the filename of the AIRSAR file.

NL (optional)
Use this optional keyword to specify a named variable that will contain the number of lines in the SIRC image.

NS (optional)
Use this optional keyword to specify a named variable that will contain the number of samples per line in SIRC image.

OFFSET (optional)
Use this optional keyword to specify a named variable that will contain the offset for the file specified by FNAME.
RANGE_KM (optional)
Use this optional keyword to specify a named variable that will contain the range or X image size in kilometers.

SLH (optional)
Use this optional keyword to specify a named variable that will contain a flag indicated whether the file needs to skip the line header. A one indicates that the line header must be skipped when processing.

TYPE (optional)
Use this optional keyword to specify an named variable that will contain the SIRC data product type for the file specified by FNAME.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MLC Quad Pole.</td>
</tr>
<tr>
<td>1</td>
<td>MLC Dual Pole HH VV</td>
</tr>
<tr>
<td>2</td>
<td>MLC Dual Pole HH HV</td>
</tr>
<tr>
<td>3</td>
<td>MLC Dual Pole VH VV</td>
</tr>
<tr>
<td>4</td>
<td>SLC Quad Pole.</td>
</tr>
<tr>
<td>5</td>
<td>SLC Dual Pole HH VV</td>
</tr>
<tr>
<td>6</td>
<td>SLC Dual Pole HH HV</td>
</tr>
<tr>
<td>7</td>
<td>SLC Dual Pole VH VV</td>
</tr>
<tr>
<td>8</td>
<td>SLC Single Pole HH</td>
</tr>
<tr>
<td>9</td>
<td>SLC Single Pole HH</td>
</tr>
</tbody>
</table>

Example

```pro
pro example_sirc_header_doit

; ; First restore all the base save files.
; envi, /restore_base_save_files

; ; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'

; ; Set the keywords
; filename = 'ndv_l.cdp'

; ; Call the "doit"
; envi_doit, 'sirc_header_doit', 
    filename=filename, ns=ns, nl=nl, 
    type=type, cancel=cancel,
```
offset=offset, band=band, az_km=az_km, $
range_km=range_km, slh=slh$
;
; Print the result
;
print, 'ns = ', ns
print, 'nl = ', nl
print, 'offset = ', offset
print, 'type = ', type
print, 'slh = ', slh
print, 'band = ', band
print, 'az_km = ', az_km
print, 'range_km = ', range_km
;
; Exit ENVI
;
envi_batch_exit
end
SIRC_MULTILOOK_DOIT

Use this program to Multilook SIRC compressed data products.

Calling Sequence

ENVI_DOIT, 'SIRC_MULTILOOK_DOIT'

Keywords

AZ_M
Use this keyword to specify the azimuth or Y image size in meters.

DIMS
Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:
• DIMS(0): Unused for this function, set to -1.
• DIMS(1): The starting X pixel. (The first pixel is number zero.)
• DIMS(2): The ending X pixel.
• DIMS(3): The starting Y pixel. (The first pixel is number zero.)
• DIMS(4): The ending Y pixel.

FNAME
Use this keyword to specify a string array of compressed data products file names for C and/or L wavelengths respectively. If a file is not used then set the array element to ''.

F_NS
Use this keyword to specify the number of samples per line in SIRC image.

F_NL
Use this keyword to specify the number of lines in the SIRC image.

LOOK
Set this keyword to specify the look factor to apply to the X and Y directions, range and azimuth respectively. LOOK is a float array of length 2.
OFFSET

Use this keyword to specify an long array of header offsets for each of the files specified by FNAME.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

RANGE_M

Use this keyword to specify the range or X image size in meters.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

SLH

Use this keyword to specify a long array indicating which files need to strip the line header. A zero indicates that line header is not present and a one indicates that the 12 byte SIRC line header must be stripped. SLH must have the same number of elements as FNAME.

TYPE

Use this keyword to specify an array of SIRC data product type for each of the files specified by FNAME.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MLC Quad Pole.</td>
</tr>
<tr>
<td>1</td>
<td>MLC Dual Pole HH VV</td>
</tr>
<tr>
<td>2</td>
<td>MLC Dual Pole HH HV</td>
</tr>
<tr>
<td>3</td>
<td>MLC Dual Pole VH VV</td>
</tr>
<tr>
<td>4</td>
<td>SLC Quad Pole.</td>
</tr>
<tr>
<td>5</td>
<td>SLC Dual Pole HH VV</td>
</tr>
<tr>
<td>6</td>
<td>SLC Dual Pole HH HV</td>
</tr>
</tbody>
</table>

Table 9-6: TYPE Keyword Values

SIRC_MULTILOOK_DOIT

ENVI Programmer's Guide
Use this keyword to specify the X offset of the input file.

**YSTART**

Use this keyword to specify the Y offset of the input file.

**Example**

```plaintext
pro example_sirc_multilook_doit

; First restore all the base save files.
; envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'

; Set the keywords
fname = 'ndv_l.cdp'

; First get necessary information about the SIR data file.
; envi_doit, 'sirc_header_doit', $
    filename=fname, ns=ns, nl=nl, $
    type=type, cancel=cancel, $
    offset=offset, band=band, az_km=az_km, $
    range_km=range_km, slh=slh

; Set the necessary keywords. Use a multilook of 10 in both the samples and lines. This will make the output
```

### Table 9-6: TYPE Keyword Values (Continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>SLC Dual Pole VH VV</td>
</tr>
<tr>
<td>8</td>
<td>SLC Single Pole HH</td>
</tr>
<tr>
<td>9</td>
<td>SLC Single Pole HH</td>
</tr>
</tbody>
</table>
dims = [-1, 0, ns-1, 0, nl-1]
az_m = (az_km * 1000.) / 10.
range_m = (range_km * 1000.) / 10.
look = [10,10]
out_name = 'testimg'

; Perform the multilook processing
envi_doit, 'sirc_multilook_doit', $
  fname=fname, f_ns=ns, f_nl=nl, $
  dims=dims, az_m=az_m, range_m=range_m, $
  offset=offset, look=look, $
  slh=slh, type=type, xstart=0, $
  ystart=0, out_name=out_name

; Exit ENVI

end
SIRC_PED_HEIGHT_DOIT

Use this program to calculate pedestal height images from a SIRC compressed data products file.

Calling Sequence

ENVI_DOIT, 'SIRC_PED_HEIGHT DOIT'

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FNAME

Use this keyword to specify a string array of compressed data products file names for C and/or L wavelengths respectively. If a file is not used then set the array element to ''.

FNS

Use this keyword to specify the number of samples per line in SIRC image.

FNL

Use this keyword to specify the number of lines in the SIRC image.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
**OUT_BNAME**

Use this keyword to specify a string array of output band names, if desired.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**OFFSET**

Use this keyword to specify a long array of header offsets for each of the files specified by FNAME.

**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**SLH**

Use this keyword to specify a long array indicating which files need to strip the line header. A zero indicates that line header is not present and a one indicates that the 12 byte SIRC line header must be stripped. SLH must have the same number of elements as FNAME.

**TYPE**

Use this keyword to specify an array of SIRC data product type for each of the files specified by FNAME.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MLC Quad Pole.</td>
</tr>
<tr>
<td>1</td>
<td>MLC Dual Pole HH VV</td>
</tr>
<tr>
<td>2</td>
<td>MLC Dual Pole HH HV</td>
</tr>
<tr>
<td>3</td>
<td>MLC Dual Pole VH VV</td>
</tr>
<tr>
<td>4</td>
<td>SLC Quad Pole.</td>
</tr>
<tr>
<td>5</td>
<td>SLC Dual Pole HH VV</td>
</tr>
<tr>
<td>6</td>
<td>SLC Dual Pole HH HV</td>
</tr>
</tbody>
</table>

*Table 9-7: TYPE Keyword Values*
Example

```plaintext
pro example_sirc_ped_height_doit
    ; First restore all the base save files.
    ; envi, /restore_base_save_files
    ; Initialize ENVI and send all errors
    ; and warnings to the file batch.txt
    ; envi_batch_init, log_file='batch.txt'
    ; Set the keywords
    ; fname = 'ndv_l.cdp'
    ; First get necessary information
    ; about the SIR data file.
    ; envi_doit, 'sirc_header_doit', $
        filename=fname, ns=ns, nl=nl, $
        type=type, cancel=cancel, $
        offset=offset, band=band, az_km=az_km, $
        range_km=range_km, slh=slh
    ;
    ; Set the DIMS keyword to calculate the
    ; pedestal height image for all spatial
    ; pixels.
    ;
    ; dims = [-1, 0, ns-1, 0, nl-1]
    ; out_name = 'testimg'
    ;
    ; Perform the processing
    ;
    ; envi_doit, 'sirc_ped_height_doit', $
        filename=fname, fns=ns, fnl=nl, $
```
dims=dims, offset=offset, slh=slh, $
  type=type, out_name=out_name, $
  r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
SIRC_PHASE_IMAGE_DOIT

Use this program to calculate phase images from a SIRC compressed data products file.

Calling Sequence

ENVI_DOIT, 'SIRC_PHASE_IMAGE_DOIT'

Keywords

DEGREE (optional)

Set this optional keyword to output the phase images in degrees. The default is radians.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FNAME

Use this keyword to specify a string array of compressed data products file names for C and/or L wavelengths. A phase image will be calculated for each file in FNAME.

FNS

Use this keyword to specify the number of samples per line in SIRC image.

FNL

Use this keyword to specify the number of lines in the SIRC image.
**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**OUT_BNAME**

Use this keyword to specify a string array of output band names, if desired.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**OFFSET**

Use this keyword to specify a long array of header offsets for each of the files specified by FNAME.

**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**SLH**

Use this keyword to specify a long array indicating which files need to strip the line header. A zero indicates that line header is not present and a one indicates that the 12 byte SIRC line header must be stripped. SLH must have the same number of elements as FNAME.

**TYPE**

Use this keyword to specify an array of SIRC data product type for each of the files specified by FNAME.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MLC Quad Pole.</td>
</tr>
<tr>
<td>1</td>
<td>MLC Dual Pole HH VV</td>
</tr>
<tr>
<td>2</td>
<td>MLC Dual Pole HH HV</td>
</tr>
<tr>
<td>3</td>
<td>MLC Dual Pole VH VV</td>
</tr>
</tbody>
</table>

*Table 9-8: TYPE Keyword Values*
Example

```fortran
pro example_sirc_phase_image_doit
    ; First restore all the base save files.
    ;
    envi, /restore_base_save_files
    ; Initialize ENVI and send all errors
    ; and warnings to the file batch.txt
    ;
    envi_batch_init, log_file='batch.txt'
    ;
    ; Set the keywords
    ;
    fname = 'ndv_l.cdp'
    ;
    ; First get necessary information
    ; about the SIR data file.
    ;
    envi_doit, 'sirc_header_doit', $
        filename=fname, ns=ns, nl=nl, $  
        type=type, cancel=cancel, $  
        offset=offset, band=band, az_km=az_km, $  
        range_km=range_km, slh=slh
    ;
    ; Set the DIMS keyword to calculate the
    ; phase image for all spatial pixels.
    ;
    dims = [-1, 0, ns-1, 0, nl-1]
    ;
    out_name = 'testimg'
```

### Table 9-8: TYPE Keyword Values (Continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>SLC Quad Pole.</td>
</tr>
<tr>
<td>5</td>
<td>SLC Dual Pole HH VV</td>
</tr>
<tr>
<td>6</td>
<td>SLC Dual Pole HH HV</td>
</tr>
<tr>
<td>7</td>
<td>SLC Dual Pole VH VV</td>
</tr>
<tr>
<td>8</td>
<td>SLC Single Pole HH</td>
</tr>
<tr>
<td>9</td>
<td>SLC Single Pole HH</td>
</tr>
</tbody>
</table>
; Perform the processing
;
envi_doit, 'sirc_phase_image_doit', $
   fname=fname, fns=ns, fnl=nl, $
   dims=dims, offset=offset, slh=slh, $
   type=type, out_name=out_name, $
   /degree, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
SIRC_POLSIG_DOIT

Use this program to calculate polarization signatures from a SIRC compressed data products file.

Calling Sequence

ENVI_DOIT, 'SIRC_POLSIG_DOIT'

Keywords

BANDS

Use this keyword to specify a two-element array of ones and zeros indicating whether the C and wavelengths were used. A value of one indicates the wavelength was used. BANDS must be two elements long regardless of the size of FNAME.

BFNAME

Use this keyword to specify a two-element string array where each element specifies C and L annotations for the header description. BFNAME must be two elements long regardless of the size of FNAME.

FNAME

Use this keyword to specify a string array of compressed data products file names for C and/or L wavelengths respectively. If a file is not used then set the array element to ''. 

FNS

Use this keyword to specify the number of samples per line in SIRC image.

FNL

Use this keyword to specify the number of lines in the SIRC image.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

OFFSET

Use this keyword to specify a long array of header offsets for each of the files specified by FNAME.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

ROI_ID

Use this keyword to specify an array of ROI IDs returned from a call to ENVI_GET_ROI_IDS. Each ID in the array will use the corresponding ROI to calculate both a co-polarization and cross-polarization image.

SLH

Use this keyword to specify a long array indicating which files need to strip the line header. A zero indicates that line header is not present and a one indicates that the 12 byte SIRC line header must be stripped. SLH must have the same number of elements as FNAME.

TYPE

Use this keyword to specify an array of SIRC data product type for each of the files specified by FNAME.

- 0: MLC Quad Pole.
- 1: MLC Dual Pole HH VV
- 2: MLC Dual Pole HH HV
- 3: MLC Dual Pole VH VV
- 4: SLC Quad Pole.
- 5: SLC Dual Pole HH VV
- 6: SLC Dual Pole HH HV
- 7: SLC Dual Pole VH VV
- 8: SLC Single Pole HH
- 9: SLC Single Pole HH
Example

forward_function envi_get_roi_ids

pro example_sirc_polsig_doit

; First restore all the base save files.
; envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
;
; Set the keywords
;
fname = 'ndv_1.cdp'
;
; First get necessary information
; about the SIR data file.
;
envi_doit, 'sirc_header_doit', $
    filename=fname, ns=ns, nl=nl, $
    type=type, cancel=cancel, $
    offset=offset, band=band, az_km=az_km, $
    range_km=range_km, slh=slh
;
; Restore the polarization signature
; ROI and get the ROI ids.
;
envi_restore_rois, 'pol_sig.roi'
roi_ids = envi_get_roi_ids(ns=ns, nl=nl)
;
; Set the DIMS keyword to calculate the
; phase image for all spatial pixels.
;
dims = [-1, 0, ns-1, 0, nl-1]
bands = [0,1,0]
out_name = 'testimg'
bfname = ['',fname,'']
;
; Perform the processing
;
envi_doit, 'sirc_polsig_doit', $
    fname=fname, fns=ns, fnl=nl, $
    offset=offset, slh=slh, type=type, $
    roi_ids=roi_ids, bfname=bfname, $
    }
bands=bands, out_name=out_name, 
  r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
SIRC_SYNTH_DOIT

Use this program to synthesize images from SIR-C compressed data products files.

Calling Sequence

ENVI_DOIT, ‘SIRC_SYNTH_DOIT’

Keywords

COMBO

Use this keyword to specify a $5 \times n$ array of ellipticity and orientation angles for each image synthesized. The format for the array is:

- $(0,i) -$ transmit ellipticity for $i$th image.
- $(1,i) -$ transmit orientation for $i$th image.
- $(2,i) -$ receive ellipticity for $i$th image.
- $(3,i) -$ receive orientation for $i$th image.
- $(4,i) -$ stokes band number 0-C, 1-L.

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FNAME

Use this keyword to specify a string array of compressed data products file names for C and/or L wavelengths respectively. If a file is not used then set the array element to ''. 
**FNS**
Use this keyword to specify the number of samples per line in SIRC image.

**FNL**
Use this keyword to specify the number of lines in the SIRC image.

**IN_MEMORY**
Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**MAX_VAL (optional)**
Use this keyword to set an optional maximum value for output data.

**MIN_VAL (optional)**
Use this keyword to set an optional minimum value for output data.

**OFFSET**
Use this keyword to specify an long array of header offsets for each of the files specified by FNAME.

**OUT_NAME**
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**OUT_DT**
Use this keyword to specify the output data type, either 1 for byte or 4 for floating point. All other output data types are invalid.

**R_FID**
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**SIGMA_ZERO**
Set this keyword to specify that the output values be converted to sigma zero, $10\times\log_{10}(data)$. 


SLH
Use this keyword to specify a long array indicating which files need to strip the line header. A zero indicates that line header is not present and a one indicates that the 12 byte SIRC line header must be stripped. SLH must have the same number of elements as FNAME.

STDMULT
Set this keyword to specify the standard deviation multiplier for byte output data types. Plus and minus the STDMULT determines the output minimum and maximum.

TOTAL_POWER
Use this keyword to specify a three element array of ones and zeros indicating whether the total power should be computed for the C and/or L wavelengths respectively. A value of one causes the synthesis of the total power image.

TYPE
Use this keyword to specify an array of SIRC data product type for each of the files specified by FNAME.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MLC Quad Pole.</td>
</tr>
<tr>
<td>1</td>
<td>MLC Dual Pole HH VV</td>
</tr>
<tr>
<td>2</td>
<td>MLC Dual Pole HH HV</td>
</tr>
<tr>
<td>3</td>
<td>MLC Dual Pole VH VV</td>
</tr>
<tr>
<td>4</td>
<td>SLC Quad Pole.</td>
</tr>
<tr>
<td>5</td>
<td>SLC Dual Pole HH VV</td>
</tr>
<tr>
<td>6</td>
<td>SLC Dual Pole HH HV</td>
</tr>
<tr>
<td>7</td>
<td>SLC Dual Pole VH VV</td>
</tr>
<tr>
<td>8</td>
<td>SLC Single Pole HH</td>
</tr>
<tr>
<td>9</td>
<td>SLC Single Pole HH</td>
</tr>
</tbody>
</table>

Table 9-9: TYPE Keyword Values
**XFAC**

Use this keyword to specify an X subsampling factor used to compute image data statistics prior to the conversion to byte. The keyword is only used when the output data type is byte.

**YFAC**

Use this keyword to specify a Y subsampling factor used to compute image data statistics prior to the conversion to byte. The keyword is only used when the output data type is byte.

**Example**

```plaintext
pro example_sirc_synth_doit

; First restore all the base save files.
envi, /restore_base_save_files

; Initialize ENVI and send all errors
; and warnings to the file batch.txt
envi_batch_init, log_file='batch.txt'

; Set the keywords
fname = 'ndv_1.cdp'

; First get necessary information
; about the SIR data file.
envi_doit, 'sirc_header_doit', $
   filename=fname, ns=ns, nl=nl, $
   type=type, cancel=cancel, $
   offset=offset, band=band, az_km=az_km, $
   range_km=range_km, slh=slh

; Set the DIMS keyword to synthesize all
; spatial pixels. Since we are only using
; the L band we need to set null values
; for the other wavelengths.
 dims = [-1, 0, ns-1, 0, nl-1]
 combo = [[0.,0.,0.,0.,1]]
 total_power = [0,1,0]
 slh = [0, slh]
```
offset = [0, offset]
fname = ['', fname]
out_name = 'testimg'

; Perform the processing
;
envi_doit, 'sirc_synth_doit', $
  fname=fname, fns=ns, fnl=nl, $
  dims=dims, offset=offset, slh=slh, $
  type=type, combo=combo, out_dt=4, $
  total_power=total_power, $
  out_name=out_name, r_fid=r_fid

; Exit ENVI
;
envi_batch_exit
end
SLICE_DOIT

Use this program to extract a spectral slice and store the result as an image.

Calling Sequence

ENVI_DOIT, ‘SLICE_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-elements array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.) This element specifies the sample for the vertical spectral slice.
- DIMS(2): The ending X pixel. (Unused for this function.)
- DIMS(3): The starting Y pixel. (The first pixel is number zero.) This element specifies the line for the horizontal spectral slice.
- DIMS(4): The ending Y pixel. (Unused for this function.)

FID

Use this keyword to specify the file ID for the open file. This is the value returned by the R_FID keyword to the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output is stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output filename for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.
**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

**R_FID**

Use this keyword to specify a named variable in which the file ID for the processed data will be returned. This file ID can be used to access the processed data.

**VERTICAL**

Set this keyword to indicate that a vertical slice is to be extracted. The extracted column is specified by dims(1). A vertical slice image has the following size: the number of samples is equal to the number of elements in POS and the number of lines is equal to the number of lines in the input image.

**HORIZONTAL**

Set this keyword to indicate that a horizontal slice is to be extracted. The extracted line is specified by dims(3). A horizontal slice image has the following size: the number of samples is equal to the number samples in the input image and the number of lines is equal to the number of elements in the POS array.
SLT2GND_DOIT

Use this program to convert from slant to ground range.

**Calling Sequence**

```
ENVI_DOIT, 'SLT2GND_DOIT'
```

**Keywords**

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**HEIGHT**

Use this keyword to specify the instrument height in meters.

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**LEFT_LOOK**

Set this keyword to specify that the instrument was left looking in the range direction.

**NR_RANGE**

Use this keyword to specify the instrument near range in meters.
OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

POS
Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

PS_OUTPUT
Use this keyword to specify the pixel size of the ground range data. PS_OUTPUT is expressed in meters.

PS_SLANT_RANGE
Use this keyword to specify the pixel size in slant range. PS_SLANT_RANGE is expressed in meters.

R_FID
Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

RESAMPLING
Use this keyword to specify the pixel resampling method. Set RESAMPLING to one of the following.
- 0 - Nearest Neighbor
- 1 - Bilinear
- 2 - Cubic Convolution

RANGE_DIR
Use this keyword to specify the range direction. If unset then the range direction is assumed to be in the samples direction. If set then the range direction is in the line direction.
Example

```plaintext
pro example_slt2gnd_doit

; First restore all the base save files.
; envi, /restore_base_save_files

; Initialize ENVI and send all errors
; and warnings to the file batch.txt
envi_batch_init, log_file='batch.txt'

; Open the input file
; envi_open_file, 'bonnrsat.img', r_fid=fid
if (fid eq -1) then begin
  envi_batch_exit
  return
endif

; Set the DIMS and POS to keywords
; to processes all spatial and all
; spectral data. Output the result
; to disk.
; envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'

; Perform the slant to ground
; correction.
; envi_doit, 'slt2gnd_doit', $
  fid=fid, pos=pos, dims=dims, $
  height=27460., nr_dist=28360, $
  ps_slant_range=8., ps_output=10.0, $
  range_dir=1, resampling=2, $
  out_name=out_name, r_fid=r_fid

; Exit ENVI
envi_batch_exit
end
```
SPECTRAL_FEATURE_DOIT

Use this program to perform spectral feature fitting.

Calling Sequence

ENVI_DOIT, ‘SPECTRAL_FEATURE_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

ENDMEM

Use this keyword to specify the endmembers array. ENDMEM is a 2-dimensional floating point array (nb, # end members).

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

M_FID

Use this keyword to specify the file ID for the mask file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.
M_POS

Use this keyword to specify the band position of the mask band. M_POS is a single long value greater than or equal to zero.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_MODE

Use this keyword to specify the output mode. OUT_MODE should be set to one of the following.

- 0 - output separate files for Scale and RMS
- 1 - output Scale/RMS to a single file.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify a one-dimension array of band positions indicating the band numbers to perform the operations on. POS is a long array ranging from 0 to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

REMOVE_CONT

Set this keyword to specify that the Continuum Removal should be run proper to Spectral Feature Fitting™.
Example

```plaintext
pro example_spectral_feature_doit
  ; First restore all the base save files.
  ;
  envi, /restore_base_save_files
  ; Initialize ENVI and send all errors
  ; and warnings to the file batch.txt
  ;
  envi_batch_init, log_file='batch.txt'
  ;
  ; Open the input file
  ;
  envi_open_file, 'mof94av.bil', r_fid=fid
  if (fid eq -1) then begin
    envi_batch_exit
    return
  endif
  ; Set the DIMS and POS to keywords
  ; to processes all spatial and all
  ; spectral data. Output the result
  ; to disk.
  ;
  envi_file_query, fid, ns=ns, nl=nl, $ nb=nb
  dims = [-1l, 0, ns-1, 0, nl-1]
  pos = lindgen(nb)
  out_name = 'testimg'
  ; Read in the endmember text file.
  ; The first column are the
  ; wavelengths and the next 19
  ; columns are the endmembers. We will
  ; use the 19 endmembers for spectral
  ; feature fitting. The endmember data
  ; must also be transposed in order to
  ; send in a (nb, # endmember) array.
  ;
  envi_read_cols, 'm94_em.asc', $
   endmem, skip=em_names, /read_skip
  endmem = transpose(endmem[1:*,*])
  out_bname = 'Fit (' + em_names[2:*] + ')
  ;
  ; Call the spectral feature fitting
  ; routine. Output the Scale/RMS image
```
; for each endmember.
;
envi_doit,'spectral_feature_doit', $
   fid=fid, pos=pos, dims=dims, $
   endmem=endmem, /remove_cont, $
   out_mode=1, out_bname=out_bname, $
   out_name=out_name, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
STRETCH_DOIT

Use this program to perform contrast stretching of image data.

Calling Sequence

ENVI_DOIT, 'STRETCH_DOIT'

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

I_MAX

Use this keyword to specify either the maximum percent stretch value or the maximum value, depending on the value of the RANGE_BY keyword.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

I_MIN

Use this keyword to specify either the minimum percent stretch value or the minimum value, depending on the value of the RANGE_BY keyword.
METHOD

Use this keyword to specify which type of stretch to perform:

- 1 - linear.
- 2 - equalize.
- 3 - gaussian.
- 4 - square root.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_DT

Set this keyword to indicate the IDL data type of the output data using the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

OUT_MAX

Use this keyword to specify the maximum output value.

OUT_MIN

Use this keyword to specify the minimum output value.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.
**RANGE_BY**

Use this keyword to indicate the type of ranging. If RANGE_BY is set to 1, the keywords I_MIN and I_MAX contain absolute minimum and maximum values. If RANGE_BY is set to zero, I_MIN and I_MAX contain percent stretch values.

**STDV**

Use this keyword to specify the standard deviation for the gaussian stretch.

**Example**

```plaintext
pro example_stretch_doit
  ; ; First restore all the base save files.
  ; envi, /restore_base_save_files
  ; ; Initialize ENVI and send all errors ; and warnings to the file batch.txt
  ; envi_batch_init, log_file='batch.txt'
  ; ; Open the input file
  ; envi_open_file, 'can_tmr.img', r_fid=fid
  if (fid eq -1) then begin
    envi_batch_exit
    return
  endif
  ; ; Set the DIMS and POS keywords
  ; to process all spatial and all ; spectral data.
  envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1l, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
  ; ; Call the stretch processing routine.
  ; We will perform a 2% stretch on the ; data and output a byte image with ; a zero to 255 range.
  ; envi_doit, 'stretch_doit', $
fid=fid, pos=pos, dims=dims, $
method=1, out_name=out_name, $
i_min=2.0, i_max=98.0, range_by=0, $
```
out_min=0, out_max=255, out_dt=1, $
  r_fid=r_fid
$
; Exit ENVI
; envi_batch_exit
end
TASCAP_DOIT

Use this program to create Tasseled Cap vegetation and soil indices.

Calling Sequence

ENVI_DOIT, ‘TASCAP_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

FILE_TYPE

Set this keyword to indicate TM or MSS data. Set FILE_TYPE to one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TM data.</td>
</tr>
<tr>
<td>1</td>
<td>MSS data.</td>
</tr>
</tbody>
</table>

Table 9-10: FILE_TYPE Keyword Values
IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1. POS must define 6 bands for TM data and 4 bands for MSS data.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_tascap_doit

; First restore all the base save files.
; envi, /restore_base_save_files
;
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
; envi_open_file, 'can_tmr.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Set the DIMS and POS to keywords
; to processes all spatial and all
; spectral data. Output the result
; to disk.
;
```
ENVI Programmer's Guide

Chapter 9: ENVI Routines

environ_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'

; Perform the Tasseled Cap vegetation
; and soil indices for the TM data.
;
envi_doit, 'tascap_doit', $
   fid=fid, pos=pos, dims=dims, $
   file_type=0, out_name=out_name, $
   r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
TEXTURE_COOCCUR_DOIT

Use this program to calculate texture co-occurrence measures for an image.

Calling Sequence

ENVI_DOIT, ‘TEXTURE_COOCCUR_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

DIRECTION

Use this keyword to specify the direction and distance between the co-occurrence kernels. DIRECTION is a two element long array of the X and Y distances and directions. For example, if DIRECTION=[2,-1] then displacement would be two pixels to the right and one line up.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
KX
Use this keyword to specify the X kernel size in pixels for calculating the texture measures. Each texture measure is calculated from the localized kernel specified by KX and KY.

KY
Use this keyword to specify the Y kernel size in pixels for calculating the texture measures. Each texture measure is calculated from the localized kernel specified by KX and KY.

METHOD
Use this keyword to specify an array of ones zero indicating which texture measure to compute. METHOD is an eight element array of integers with the following definitions:

- METHOD(0) - Compute the co-occurrence mean.
- METHOD(1) - Compute the co-occurrence variance.
- METHOD(2) - Compute the co-occurrence homogeneity.
- METHOD(3) - Compute the co-occurrence contrast.
- METHOD(4) - Compute the co-occurrence dissimilarity.
- METHOD(5) - Compute the co-occurrence entropy.
- METHOD(6) - Compute the co-occurrence second moment.
- METHOD(7) - Compute the co-occurrence correlation.

OUT_BNAME
Use this keyword to specify a string array of output band names, if desired.

OUT_NAME
Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS
Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands - 1.
**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**Example**

```plaintext
pro example_texture_cooccur_doit

; First restore all the base save files.
envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
envi_batch_init, log_file='batch.txt'

; Open the input file
envi_open_file, 'can_tmr.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif

; Set the DIMS and POS to keywords to processes all spatial and all spectral data. Output the result to disk.
envi_file_query, fid, ns=ns, nl=nl1, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
method = lonarr(8) + 1
direction = [2,2]
out_name = 'testimg'

; Perform the texture cooccurrence calculation.
envi_doit, 'texture_cooccur_doit', fid=fid, pos=pos, dims=dims, method=method, kx=5, ky=5, direction=direction, out_name=out_name, r_fid=r_fid

; Exit ENVI
```
;  
envi_batch_exit  
end
TEXTURE_STATS_DOIT

Use this program to calculate texture measure for an image.

Calling Sequence

ENVI_DOIT, ‘TEXTURE_STATS_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

KX

Use this keyword to specify the X kernel size in pixels for calculating the texture measures. Each texture measure is calculated from the localized kernel specified by KX and KY.
KY

Use this keyword to specify the Y kernel size in pixels for calculating the texture measures. Each texture measure is calculated from the localized kernel specified by KX and KY.

METHOD

Use this keyword to specify an array of ones zero indicating which texture measure to compute. METHOD is a five element array of integers with the following definitions:

- METHOD(0) - Compute the kernel data range.
- METHOD(1) - Compute the kernel mean.
- METHOD(2) - Compute the kernel variance.
- METHOD(3) - Compute the kernel entropy.
- METHOD(4) - Compute the kernel skewness

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

```plaintext
pro example_texture_stats_doit
;
; First restore all the base save files.
;
envi, /restore_base_save_files
;```
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
;
; Open the input file
;
envi_open_file, 'bhtmref.img', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;
; Set the DIMS keyword to
; process all spatial and use
; the POS keyword to calculate
; the texture measures only for
; the first band (band zero).
; Use a 5x5 kernel to calculate
; all texture measures (data range,
; mean, variance, entropy, skewness).
; The results will be save to disk.
;
envi_file_query, fid, ns=ns, nl=nl, nb=nb
Dims = [-1l, 0, ns-1, 0, nl-1]
pos = [0]
out_name = 'testimg'
out_bname = ['Data Range', 'Mean', $
    'Variance', 'Entropy', 'Skewness']
method = [1,1,1,1,1]
;
; Call the texture processing
; routine.
;
envi_doit, 'texture_stats_doit', $
    fid=fid, pos=pos, dims=dims, $
    kx=5, ky=5, method=method, $
    out_name=out_name, r_fid=r_fid, $
    out_bname=out_bname
;
; Exit ENVI
;
envi_batch_exit
end
TIMS_CAL_DOIT

Use this program to calibrate TIMS data to radiance. This routine uses the calibration information at the start of each line to calibrate the data.

Calling Sequence

ENVI_DOIT, ‘TIMS_CAL_DOIT’

Keywords

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.
**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

**R_FID (optional)**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**REF_SMOOTH**

Use this keyword to specify the width of the reference data moving average window. REF_SMOOTH lines are averaged together and used for the calibration. REF_SMOOTH is a single long value.

**Example**

```plaintext
pro example_tims_cal_doit

; First restore all the base save files.
envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
envi_batch_init, log_file='batch.txt'

; Open the input file
envi_open_file, 'tims_data.raw', r_fid=fid
if (fid eq -1) then begin
    envi_batch_exit
    return
endif

; Set the DIMS and POS to keywords to processes all spatial and all spectral data. Output the result to disk.
envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'
```

TIMS_CAL_DOIT ENVI Programmer's Guide
; Perform the TIMS Calibration.

; envi_doit, 'tims_cal_doit', $
  fid=fid, pos=pos, dims=dims, $
  ref_smooth=20, out_name=out_name, $
  r_fid=r_fid

; Exit ENVI

end

See Also

EMITTANCE_CALC_DOIT
TMCAL_DOIT

Use this program to calibrate Landsat TM data to radiance or reflectance using pre-launch characteristics.

Calling Sequence

ENVI_DOIT, ‘TMCAL_DOIT’

Keywords

BIAS

Use this keyword to specify the calibration bias values for Landsat 7. This keyword is not used for other satellites. BIAS is used with GAIN to convert the data.

\[ \text{result} = \text{Input} \times \text{GAIN} + \text{BIAS} \]

CAL_TYPE

Use this keyword to specify the calibration type. Set CAL_TYPE equal to zero to indicate Radiance or to one to indicate Reflectance.

DATE

Use this keyword to specify the calibration numbers to use, depending on the date the data was collected. Choose one of the following (this keyword is not used for Landsat 7, SAT=7):

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Before August 1983</td>
</tr>
<tr>
<td>1</td>
<td>Before January 15, 1984</td>
</tr>
<tr>
<td>2</td>
<td>After January 15, 1984</td>
</tr>
</tbody>
</table>

Table 9-11: DATE Keyword Values

DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
• DIMS(1): The starting X pixel. (The first pixel is number zero.)
• DIMS(2): The ending X pixel.
• DIMS(3): The starting Y pixel. (The first pixel is number zero.)
• DIMS(4): The ending Y pixel.

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**GAIN**

Use this keyword to specify the calibration gain values for Landsat 7. This keyword is not used for other satellites. GAIN is used with BIAS to convert the data.

\[
\text{result} = \text{Input} * \text{GAIN} + \text{BIAS}.
\]

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

**OUT_BNAME**

Use this keyword to specify a string array of output band names, if desired.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.
SAT

Set this keyword to one of the following values to indicate the satellite type:

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Landsat 4</td>
</tr>
<tr>
<td>5</td>
<td>Landsat 5</td>
</tr>
<tr>
<td>7</td>
<td>Landsat 7</td>
</tr>
</tbody>
</table>

Table 9-12: SAT Keyword Values

SUN_ANGLE

Use this keyword to specify a floating-point value between 0.0 and 90.0 corresponding to the sun elevation angle. Sun elevation is only used for reflectance calibration.

Example

```plaintext
pro example_tmcal_doit
 ;
 ; First restore all the base save files.
 ;
 envi, /restore_base_save_files
 ;
 ; Initialize ENVI and send all errors and warnings to the file batch.txt
 ;
 envi_batch_init, log_file='batch.txt'
 ;
 ; Open the input file
 ;
 envi_open_file, 'can_tmr.img', r_fid=fid
 if (fid eq -1) then begin
   envi_batch_exit
   return
 endif
 ;
 ; Set the DIMS and POS to keywords to processes all spatial and all spectral data. Output the result to disk.
 ;
 envi_file_query, fid, ns=ns, nl=nl, nb=nb
```
dims = [-1, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
bands_present = [0,1,2,3,4,6]
out_name = 'testimg'
;
; Perform the TM Calibration
;
envi_doit, 'tmcal_doit', $
   fid=fid, pos=pos, dims=dims, $
   bands_present=bands_present, $
   sat=4, cal_type=1, date=0, $
   sun_angle=38.0, out_name=out_name, $
   r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
TOPO_DOIT

Use this program to perform topographic modeling of a DEM image file.

Calling Sequence

ENVI_DOIT, ‘TOPO_DOIT’

Keywords

AZIMUTH

Use this keyword to specify the sun azimuth for the purpose of sun shading.

BPTR (optional)

Use this keyword to specify the array of images to generate. Setting the elements of the BPTR array to the values shown below will generate that image. For example, to generate only the Slope and Shaded Relief images, set BPTR = [0,2].

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slope</td>
</tr>
<tr>
<td>1</td>
<td>Aspect</td>
</tr>
<tr>
<td>2</td>
<td>Shaded Relief</td>
</tr>
<tr>
<td>3</td>
<td>Profile Convexity,</td>
</tr>
<tr>
<td>4</td>
<td>Plan Convexity</td>
</tr>
<tr>
<td>5</td>
<td>Longitudinal Convexity,</td>
</tr>
<tr>
<td>6</td>
<td>Cross Sectional Convexity</td>
</tr>
<tr>
<td>7</td>
<td>Minimum Curvature</td>
</tr>
<tr>
<td>8</td>
<td>Maximum Curvature</td>
</tr>
<tr>
<td>9</td>
<td>RMS</td>
</tr>
</tbody>
</table>

Table 9-13: BPTR Keyword Values

If this keyword is unspecified then all images will be generated.
DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

ELEVATION

Use this keyword to specify the sun elevation for the purpose of sun shading.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.

OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

PIXEL_SIZE

Use this keyword to specify a two-element array containing the X and Y pixel sizes, respectively. The pixel size should be in the same units as the DEM.
POS

Use this keyword to specify a single band position, indicating the band number on which to perform the operation. POS is a one element long integer array, the range on the value is from zero to the number of bands-1.

R_FID

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

Example

forward_function envi_get_projection
pro example_topo_doit

; First restore all the base save files.
; envi, /restore_base_save_files
; Initialize ENVI and send all errors
; and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
; Open the input file
; envi_open_file, 'bhdemsub.img', r_fid=fid
if (fid eq -1) then begin
  envi_batch_exit
  return
endif

envi_file_query, fid, ns=ns, nl=nl
dims = [-1l, 0, ns-1, 0, nl-1]
pos = [0]
bptr = [0, 1, 2]
out_bname = ['Slope', 'Aspect', $
'Shaded Relief']
proj = envi_get_projection(fid=fid, $
  pixel_size=pixel_size)
out_name = 'testimg'
;
; Call the topographic processing
; routine. Use a sun elevation of
; 67 degrees and an azimuth of 23
; degrees.
;
envi_doit, 'topo_doit', $
  fid=fid, pos=pos, dims=dims, $
  azimuth=23.0, elevation=67.0, $
  bptr=bptr, out_name=out_name, $
  out_bname=out_bname, $
  pixel_size=pixel_size, r_fid=r_fid
;
; Exit ENVI
;
envi_batch_exit
end
TOPOFEATURE_DOIT

Use this program to create a topographic feature classification image. Each pixel is classified into one of the following terrain types or morphometric features: peak, ridge, pass, plane, channel, or pit as controlled by the setting of CPTR.

Calling Sequence

ENVI_DOIT, 'TOPO_FEATURE_DOIT'

Keywords

CLASS_NAMES

Use this keyword to specify names for each output class. CLASS_NAMES is an array of strings with num_classes+1 elements. Remember to set the zero class to “Unclassified”.

CONVEX_TOL

Use this keyword to specify the floating-point or double-precision value for the curvature tolerance. CONVEX_TOL and SLOPE_TOL in addition to the topographic modeling values determine how a pixel is classified.

CPTR

Use this keyword to specify the output classes, peak, ridge, pass, plane, channel, or pit, to compute. CPTR is an long array equal of values for each of the output classes to compute. The possible values and the corresponding classes for CPTR are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Peak</td>
</tr>
<tr>
<td>1</td>
<td>Ridge</td>
</tr>
<tr>
<td>2</td>
<td>Pass</td>
</tr>
<tr>
<td>3</td>
<td>Plane</td>
</tr>
<tr>
<td>4</td>
<td>Channel</td>
</tr>
<tr>
<td>5</td>
<td>Pit</td>
</tr>
</tbody>
</table>

Table 9-14: CPTR Keyword Values
DIMS

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

FID

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

IN_MEMORY

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.

KERNEL_SIZE

Use this keyword to specify a kernel size for the topographic features. KERNEL_SIZE is a single scalar that set both the X and Y kernel size. KERNEL_SIZE determines the local surface fit size used to calculate topographic modeling parameters for feature extraction.

LOOKUP

Use this keyword to specify an array specifying the color tables for the classification image. Each output class can have a unique color triple [r, g, b], LOOKUP is a byte array of size (3, num_classes+1). Remember that class zero must also have a color triplet (commonly black [0,0,0]).

OUT_BNAME

Use this keyword to specify a string array of output band names, if desired.
OUT_NAME

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

PIXEL_SIZE

Use this optional keyword to specify the pixel size of the image. PIXEL_SIZE is a two element floating or double precision array specifying the X and Y pixel size respectively.

POS

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

R_FID (optional)

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

SLOPE_TOL

Use this keyword to specify the floating-point or double-precision value for the slope tolerance. SLOPE_TOL and CONVEX_TOL in addition to the topographic modeling values determine how a pixel is classified.

Example

```plaintext
forward_function envi_get_projection

pro example_topo_feature_doit
  ;
  ; First restore all the base save files.
  ;
  envi, /restore_base_save_files
  ;
  ; Initialize ENVI and send all errors
  ; and warnings to the file batch.txt
  ;
  envi_batch_init, log_file='batch.txt'
  ;
  ; Open the input file
  ;
  envi_open_file, 'bhdemsub.img', r_fid=fid
  if (fid eq -1) then begin
```
    envi_batch_exit
    return
endif

; Set the DIMS keyword to process
; all the spatial data and set
; the POS keyword to use the
; first band from the file.
; Since the data file is
; georeferenced we can use the
; pixel size from the projection.
; The result will be saved to disk.
; Use a 7x7 kernel with a slope
; tolerance of 1.0 and a convex
; tolerance of 0.1
;
    envi_file_query, fid, ns=ns, nl=nl, $ 
        sname=sname
    dims = [-1l, 0, ns-1, 0, nl-1]
    pos = [0]
    out_name = 'testimg'
    proj = envi_get_projection(fid=fid, $ 
        pixel_size=pixel_size)
    kernel_size = 7L
    slope_tol = 1.0
    convex_tol = .1
;
; We will classify all feature (peak, 
; ridge, pass, plane, channel and pit)
; names - CPTR keyword. Set the output
; band name, class names and class
; colors.
;
    cptr = bytarr(6) + 1b
    out_bname = 'Topographic Features (' + $ 
        sname + ')
    class_names = ['Unclassified', 'Peak', $ 
        'Ridge', 'Pass', 'Plane', 'Channel', $ 
        'Pit']
    lookup = bytarr(3,7)
    for i=0, 6 do begin
        envi_get_rgb_triplets, i+2, r, g, b
        lookup[0,i] = [r,g,b]
    endfor
;
; Call the topographic feature
; classification routine.
;
    envi_doit,'topo_feature_doit', $
See Also

TOPO_DOIT
UNMIX_DOIT

Use this program to perform Linear Spectral Unmixing.

**Calling Sequence**

ENVI_DOIT, ‘UNMIX_DOIT’

**Keywords**

**DIMS**

Use this keyword to specify the spatial dimensions on which to perform the operation. DIMS is a five-element array of long integers with the following definitions:

- DIMS(0): Unused for this function, set to -1.
- DIMS(1): The starting X pixel. (The first pixel is number zero.)
- DIMS(2): The ending X pixel.
- DIMS(3): The starting Y pixel. (The first pixel is number zero.)
- DIMS(4): The ending Y pixel.

**ENDMEM**

Use this keyword to specify a floating-point array of the end members to unmix with. This array must already be resampled to the correct wavelength to the data. The dimensions of ENDMEM are (nb, num_endmem).

**FID**

Use this keyword to specify the file ID for the open file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**IN_MEMORY**

Set this keyword to specify that output should be stored in memory. If IN_MEMORY is not set, output will be stored on disk.
**M_FID**

Use this keyword to specify the file ID for the mask file. This is the value returned from the keyword R_FID in the ENVI_OPEN_FILE procedure. FID is a long integer with a value greater than zero. An invalid file ID is specified as -1.

**M_POS**

Use this keyword to specify the band position of the mask band. M_POS is a single long value greater than or equal to zero.

**OUT_NAME**

Use this keyword to specify an output file name for the resulting data. If the keyword IN_MEMORY is set, this keyword is not needed.

**OUT_BNAME**

Use this keyword to specify a string array of output band names, if desired.

**POS**

Use this keyword to specify an array of band positions, indicating the band numbers on which to perform the operation. POS is an array of long integers, ranging from zero to the number of bands-1.

**R_FID**

Use this keyword to specify a named variable that will contain the file ID for the processed data. This file ID can be used to access the processed data.

**Weight (optional)**

Use this optional keyword to specify a weight when applying a unit sum constraint. WEIGHT is a floating point value greater than zero.

**Example**

```pro
pro example_unmix_doit

; First restore all the base save files.
; envi, /restore_base_save_files

; Initialize ENVI and send all errors and warnings to the file batch.txt
; envi_batch_init, log_file='batch.txt'
```
ENVI Routines

Chapter 9: ENVI Routines

ENVI Programmer’s Guide

; Open the input file
envi_open_file, 'mof94av.bil', r_fid=fid
if (fid eq -1) then begin
  envi_batch_exit
  return
endif

; Set the DIMS and POS to keywords
; to process all spatial and all spectral data. Output the result
; to disk.
envi_file_query, fid, ns=ns, nl=nl, nb=nb
dims = [-1l, 0, ns-1, 0, nl-1]
pos = lindgen(nb)
out_name = 'testimg'

; Read in the endmember text file.
; The first column are the wavelengths and the next 19 columns are the endmembers. We will use the 19 endmembers for unmixing.
; The endmember data must also be transposed in order to send in a (nb, # endmember) array.
envi_read_cols, 'm94_em.asc', endmem, skip=em_names, /read_skip
endmem = transpose(endmem[1:*,*])
out_bname = ['Unmix EM:' + em_names[2:*], 'RMS Error']

; Call the Unmixing processing routine.
envi_doit,'unmix_doit', fid=fid, pos=pos, dims=dims, endmem=endmem, out_name=out_name, out_bname=out_bname, in_memory=0, r_fid=r_fid

; Exit ENVI
envi_batch_exit
end
VAX_IEEE_DOIT

Use this program to perform VAX IEEE floating point conversion.

Calling Sequence

ENVI_DOIT, ‘VAX_IEEE_DOIT’

Keywords

IN_NAME

Use this keyword to specify an input file name for the VAX IEEE data.

OUT_NAME

Use this keyword to specify an output file name for the resulting data.

OFFSET

Use this keyword to specify a byte offset to the start of the VAX IEEE floating point data.

COPY_HEADER

Set this keyword to specify that the header should be copied to the new output file.

Example

```pro
pro example_vax_ieee_doit
    ; First restore all the base save files.
    ; envi, /restore_base_save_files
    ; Initialize ENVI and send all errors and warnings to the file batch.txt
    ; envi_batch_init, log_file='batch.txt'
    ; Set the input VAX filename.
    ; offset = 0L
    in_name = 'vaxdata'
    out_name = 'testimg'
    ; Perform the VAX IEEE conversion.
```

```
; envi_doit, 'vax_ieee_doit', $
   in_name=in_name, out_name=out_name, $
   offset=offset
;
; Exit ENVI
;
envi_batch_exit
end
WIDGET_AUTO_BASE

The WIDGET_AUTO_BASE function is used to create the top level base widget for any auto-managed ENVI widget.

The returned value of this function is the widget ID of the newly-created base.

Calling Sequence

Result = WIDGET_AUTO_BASE()

Keywords

GROUP (optional)

The widget ID of an existing widget that serves as “group leader” for the newly-created widget. When a group leader is killed, for any reason, all widgets in the group are also destroyed.

A given widget can be in more than one group. The WIDGET_CONTROL procedure can be used to add additional group associations to a widget. It is not possible to remove a widget from an existing group.

TITLE (optional)

A string containing the title to be used for the top-level widget.

XBIG (optional)

Set this optional keyword if the widget will be large in the horizontal direction.

YBIG (optional)

Set this optional keyword if the widget will be large in the vertical direction.

Example

; *******************************************************
; This routine is an example of how to select multiple
; ROIs using a compound widget.
;
; For more information see the ENVI Programmer's Guide.
; *******************************************************
pro roi_multi_sel
envi_select, title='Input Filename', fid=fid
if (fid eq -1) then return
roi_ids = envi_get_roi_ids(fid=fid, roi_names=roi_names)
if (roi_ids[0] eq -1) then begin
  print, 'No regions associated with the selected file'
  return
endif
; Compound widget for ROI selection
base = widget_auto_base(title='ROI Selection')
wm = widget_multi(base, list=roi_names, uvalue='list', /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
ptr = where(result.list eq 1)
print, roi_names[ptr]
print, roi_ids[ptr]
end
WIDGET_BASE

The WIDGET_BASE function is used to create base widgets. Base widgets serve as containers for other widgets.

The returned value of this function is the widget ID of the newly-created base.

Calling Sequence

Result = WIDGET_BASE([Parent])

Note

This function has been partially documented here for the convenience of the user, for a complete list of arguments and keywords for WIDGET_BASE see the IDL Reference Guide.

Arguments

Parent

The widget ID of the parent widget. To create a top-level base, omit the Parent argument.

Keywords

COLUMN

If this keyword is included, the base lays out its children in columns. The value of this keyword specifies the number of columns to be used. When one column is filled, a new one is started.

Note that the standard base widget does not impose any placement constraints on its child widgets. However, laying out widgets using the XSIZE, YSIZE, XOFFSET, and YOFFSET keywords can be both tedious and error-prone. Also, if you want your widget application to display properly on different platforms, you should use the COLUMN and ROW keywords to influence child widget layouts instead of explicitly formatting your interfaces.

EVENT_FUNC

A string containing the name of a function to be called by the WIDGET_EVENT function when an event arrives from a widget in the widget hierarchy rooted at the newly-created widget.
EVENT_PRO
A string containing the name of a procedure to be called by the WIDGET_EVENT function when an event arrives from a widget in the widget hierarchy rooted at the newly-created widget.

EXCLUSIVE
Set this keyword to specify that the base can have only button-widget children and that only one button can be set at a time. These buttons, unlike normal button widgets, have two states-set and unset.

When one exclusive button is pressed, any other exclusive buttons (in the same base) that are currently set are automatically released. Hence, only one button can ever be set at one time.

FRAME
The value of this keyword specifies the width of a frame (in pixels) to be drawn around the borders of the widget. Note that this keyword is only a hint to the toolkit, and may be ignored in some instances.

FUNC_GET_VALUE
A string containing the name of a function to be called when the GET_VALUE keyword to the WIDGET_CONTROL procedure is called for this widget. Using this technique allows you to change the value that should be returned for a widget. Compound widgets use this ability to define their values transparently to the user.

GROUP_LEADER
The widget ID of an existing widget that serves as “group leader” for the newly-created widget. When a group leader is killed, for any reason, all widgets in the group are also destroyed.

A given widget can be in more than one group. The WIDGET_CONTROL procedure can be used to add additional group associations to a widget. It is not possible to remove a widget from an existing group.

MAP
Once a widget hierarchy has been realized, it can be mapped (visible) or unmapped (invisible). This keyword specifies the initial map state for the given base and its descendants. Specifying a non-zero value indicates that the base should be mapped when realized (the default). A zero value indicates that the base should be unmapped...
initially. After the base is realized, its map state can be altered using the MAP keyword to the WIDGET_CONTROL procedure.

**MBAR**

Set this keyword to a named variable to cause a menu bar to be placed at the top of the base (the base must be a top-level base). The menu bar is itself a special kind of base widget that can only have buttons as children. Upon return, the named variable contains the widget ID of the new menu bar base. This widget ID can then be used to fill the menu bar with pulldown menus.

**NO_COPY**

Usually, when the SET_UVALUE, GET_UVALUE, or SEND_EVENT keywords are used, the source variable memory is copied. When the NO_COPY keyword is set, the source variable dynamic memory is used by the destination, without copying. This feature can be used by compound widgets to obtain state information from a UVALUE without all the memory copying that would otherwise occur.

**NONEXCLUSIVE**

Set this keyword to specify that the base can only have button widget children. These buttons, unlike normal button widgets, have two states-set and unset. Non-exclusive bases allow any number of the toggle buttons to be set at one time.

**PRO_SET_VALUE**

A string containing the name of a procedure to be called when the SET_VALUE keyword to the WIDGET_CONTROL procedure is called for this widget. Using this technique allows you to designate a routine that sets the value for a widget. Compound widgets use this ability to define their values transparently to the user.

**ROW**

If this keyword is included, the base lays out its children in rows. The value of this keyword specifies the number of rows to be used. When one row is filled, a new one is started.

This keyword is similar to the COLUMN keyword described above.

**TITLE**

A string containing the title to be used for the widget. Base widgets use the title only if they are top-level widgets.
UVALUE

The “user value” to be assigned to the widget.

Each widget can contain a user-specified value of any data type and organization. This value is not used by the widget in any way, but exists entirely for the convenience of the IDL programmer. This keyword allows you to set this value when the widget is first created.

If UVALUE is not present, the widget's initial user value is undefined.

The user value for a widget can be accessed and modified at any time by using the GET_UVALUE and SET_UVALUE keywords to the WIDGET_CONTROL procedure.

XOFFSET

The horizontal offset of the widget (in pixels) relative to its parent.

Specifying an offset relative to a row or column major base widget does not work because those widgets enforce their own layout policies. This keyword is primarily of use relative to a plain base widget. Note that it is best to avoid using this style of widget layout.

XSIZE

The width of the widget (in pixels for most widgets, in characters for text widgets).

Most widgets attempt to size themselves to fit the situation. However, if the desired effect is not produced, use this keyword to override it. This keyword is only a “hint” to the toolkit and may be ignored in some situations.

YOFFSET

The vertical offset of the widget (in pixels) relative to its parent. This offset is specified relative to the upper left corner of the parent widget.

Specifying an offset relative to a row or column major base widget does not work because those widgets enforce their own layout policies. This keyword is primarily of use relative to a plain base widget. Note that it is best to avoid using this style of widget layout.

YSIZE

The height of the widget (in pixels for most widgets, in characters for text widgets, in number of visible items for list widgets).
Most widgets attempt to size themselves to fit the situation. However, if the desired effect is not produced, use this keyword to override it. This keyword is only a “hint” to the toolkit and may be ignored in some situations.

**Widget Events**

Top-level widget bases return the following event structure only when they are resized by the user and the base was created with the TLB_SIZE_EVENTS keyword set:

\[
(WIDGET\_BASE, \ ID:0L, \ TOP:0L, \ HANDLER:0L, \ X:0, \ Y:0)
\]

ID is the widget ID of the base generating the event. TOP is the widget ID of the top level widget containing ID. HANDLER contains the widget ID of the widget associated with the handler routine. The X and Y fields return the new width of the base, not including any frame provided by the window manager.

**Note**

For a complete list of arguments and keywords for WIDGET_BASE see the *IDL Reference Guide.*
The WIDGET_CONTROL procedure is used to realize, manage, and destroy widget hierarchies. It is often used to change the default behavior or appearance of previously-realized widgets.

**Calling Sequence**

```
WIDGET_CONTROL [, Widget_ID]
```

**Note**

This function has been partially documented here for the convenience of the user, for a complete list of arguments and keywords for WIDGET_CONTROL see the *IDL Reference Guide*.

**Arguments**

**Widget_ID**

The widget ID of the widget to be manipulated. This argument can be omitted when used with the RESET keyword, but is required for all other operations.

**Keywords**

**DESTROY**

Set this keyword to destroy the widget and any child widgets in its hierarchy. Any further attempts to use the IDs for these widgets will cause an error.

**EVENT_FUNC**

A string containing the name of a function to be called by the WIDGET_EVENT function when an event arrives from a widget in the widget hierarchy given by Widget_ID.

This keyword overwrites any event routine supplied by previous uses of the EVENT_FUNC or EVENT_PRO keywords. To specify no event routine, set this keyword to a null string ('').
EVENT_PRO

A string containing the name of a procedure to be called by the WIDGET_EVENT function when an event arrives from a widget in the widget hierarchy given by Widget_ID.

This keyword overwrites any event routine supplied by previous uses of the EVENT_FUNC or EVENT_PRO keywords. To specify no event routine, set this keyword to a null string ("").

FUNC_GET_VALUE

A string containing the name of a function to be called when the GET_VALUE keyword to the WIDGET_CONTROL procedure is called for this widget. Using this technique allows you to change the value that should be returned for a widget. Compound widgets use this ability to define their values transparently to the user.

GET_UVALUE

Set this keyword to a named variable to contain the current user value of the widget.

Each widget can contain a user set value of any data type and organization. This value is not used by the widget in any way, and exists entirely for the convenience of the IDL programmer. This keyword allows you to obtain the current user value.

The user value of a widget can be set with the SET_UVALUE keyword to this routine, or with the UVALUE keyword to the routine that created it.

To improve the efficiency of the data transfer, consider using the NO_COPY keyword (described below) with GET_UVALUE.

GET_VALUE

Set this keyword to a named variable to contain the current value of the widget. The type of value returned depends on the widget type:

- Button: If the button label is text, it is returned as a string. Attempts to obtain the value of a button with a bitmap label is an error.
- Draw: The IDL window ID for the drawing area is returned. This ID is used with procedures such as WSET, WSHOW, etc., to direct graphics to the widget. The window ID is assigned to drawing area widgets at the time they are realized. If the widget has not yet been realized, a value of -1 is returned.
- Label: The label text is returned as a string.
- Slider: The current value of the slider is returned as an integer.
- Text: The current contents of the text widget are returned as a string array. If the USE_TEXT_SELECT keyword is also specified, only the contents of the current selection are returned.

- Widget types not listed above do not return a value. Attempting to retrieve the value of such a widget causes an error.

The value of a widget can be set with the SET_VALUE keyword to this routine, or with the VALUE keyword to the routine that created it.

**GROUP_LEADER**

The widget ID of an existing widget that serves as “group leader” for the newly-created widget. When a group leader is killed, for any reason, all widgets in the group are also destroyed.

A given widget can be in more than one group. The WIDGET_CONTROL procedure can be used to add additional group associations to a widget. It is not possible to remove a widget from an existing group.

**HOURGLASS**

Set this keyword to turn on an “hourglass-shaped” cursor for all IDL widgets and graphics windows. The hourglass remains in place until the WIDGET_EVENT function attempts to process the next event. Then the previous cursor is reinstated. If an application starts a time-intensive calculation inside an event-handling routine, the hourglass cursor should be used to indicate that the system is not currently responding to events.

**ICONIFY**

Set this keyword to a non-zero value to cause the specified widget to become iconified. Set this keyword to zero to open an iconified widget.

**INPUT_FOCUS**

If Widget_ID is a text widget, you can set this keyword to cause the widget to receive the keyboard focus. If Widget_ID is a button widget, set this keyword to position the mouse pointer over the button (on Motif), or set the focus to the button so that it can be “pushed” with the spaceport (on Windows). You cannot set the input focus to a button in IDL for Macintosh. If Widget_ID is a draw widget, set this keyword to give it the focus in IDL for Macintosh; this allows you to print from the draw widget. This keyword has no effect for other widget types.
MAP

Set this keyword to zero to unmap the widget hierarchy rooted at the widget specified by Widget_ID. The hierarchy disappears from the screen, but still exists.

The mapping operation applies only to base widgets. If the specified widget is not a base, IDL searches upward in the widget hierarchy until it finds the closest base widget. The map operation is applied to that base.

Set MAP to a nonzero value to re-map the widget hierarchy and make it visible. Normally, the widget is automatically mapped when it is realized, so use of the MAP keyword is not required.

NO_COPY

Usually, when the SET_UVALUE, GET_UVALUE, or SEND_EVENT keywords are used, the source variable memory is copied. When the NO_COPY keyword is set, the source variable dynamic memory is used by the destination, without copying. This feature can be used by compound widgets to obtain state information from a UVALUE without all the memory copying that would otherwise occur.

Note that when the NO_COPY keyword is set, data is taken away from the source and attached directly to the destination. This feature can be used to move data very efficiently. However, it has the side effect of causing the source variable to become undefined.

PRO_SET_VALUE

A string containing the name of a procedure to be called when the SET_VALUE keyword to the WIDGET_CONTROL procedure is called for this widget. Using this technique allows you to designate a routine that sets the value for a widget. Compound widgets use this ability to define their values transparently to the user.

REALIZE

If set, the widget hierarchy is realized. Until the realization step, the widget hierarchy exists only within IDL. Realization is the step of actually creating the widgets on the screen (and mapping them if necessary).

When a previously-realized widget gets a new child widget, the new child is automatically realized.

SENSITIVE

When a widget is sensitive, it has normal appearance and can receive user input. For instance, a sensitive button widget can be activated by moving the mouse cursor over
it and pressing a mouse button. When a widget is insensitive, it indicates the fact by changing its appearance, and ignores any input directed at it. If SENSITIVE is zero, the widget hierarchy becomes insensitive. If nonzero, it becomes sensitive.

Sensitivity can be used to control when a user is allowed to manipulate a widget. It should be noted that some widgets do not change their appearance when they are made insensitive, and simply cease generating events.

**SET_UVALUE**

Each widget can contain a user-set value. This value is not used by IDL in any way, and exists entirely for the convenience of the IDL programmer. This keyword allows you to set this value.

To improve the efficiency of the data transfer, consider using the NO_COPY keyword with SET_UVALUE.

**SET_VALUE**

Sets the value of the specified widget. The meaning of the value differs between widget types:

- **Button**: The label to be used for the button. This value can be either a scalar string, or a 2D byte array containing a bitmap.
- **Label**: The text to be displayed by the label widget.
- **List**: Set the selections for the list widget (string or string array).
- **Slider**: Sets the current position of the slider (integer).
- **Text**: Set the text to be displayed. If the APPEND keyword is also specified, the text is appended to the current contents instead of completely replacing it (string or string array). If the USE_TEXT_SELECT keyword is specified, the new string replaces only the currently-selected text in the text widget.

- Widget types not listed above do not allow the setting of a value. Attempting to set the value of such a widget causes an error.

The value of a widget can also be set with the VALUE keyword to the routine that created it.

**XOFFSET**

Set this keyword to an integer value that specifies the widget's new horizontal offset, in pixels. Attempting to change the offset of a widget that is the child of a
ROW or COLUMN base or a widget that is part of a menu bar or pulldown menu causes an error.

**XSIZE**

Set this keyword to an integer value that represents the widget's new horizontal size in pixels for most widgets, but in characters for text or list widgets. For most non-scrollable widgets, this size is the same as the “screen size” that can be set using the SCR_XSIZE keyword. For scrollable widgets (e.g., scrolling bases and scrolling draw widgets), this keyword adjusts the viewport size. Use the DRAW_XSIZE keyword to change the width of the drawing area in scrolling draw widgets. Attempting to resize a widget that is part of a menubar or pulldown menu causes an error.

**YOFFSET**

Set this keyword to an integer value that specifies the widget's new vertical offset, in pixels. Attempting to change the offset of a widget that is the child of a ROW or COLUMN base or a widget that is part of a menu bar or pulldown menu causes an error.

**YSIZE**

Set this keyword to an integer value that represents the widget's new vertical size in pixels for most widgets, but in characters for text or list widgets. For most non-scrollable widgets, this size is the same as the “screen size” that can be set using the SCR_YSIZE keyword. For scrollable widgets (e.g., scrolling bases and scrolling draw widgets), this keyword adjusts the viewport size. Use the DRAW_YSIZE keyword to change the height of the drawing area in scrolling draw widgets. Attempting to resize a widget that is part of a menu bar or pulldown menu causes an error.

**Note**

For a complete list of arguments and keywords for WIDGET_CONTROL see the *IDL Reference Guide*.

**See Also**

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_BASE
WIDGET_EDIT

The WIDGET_EDIT compound widget is used to edit multiple values from lists. The function returns the base ID of the widget.

Calling Sequence

\[ \text{Result} = \text{WIDGET_EDIT}(\text{Base}) \]

Arguments

Base

The widget ID of the base widget.

Keywords

AUTO_MANAGE (optional)

Use this optional keyword to specify how ENVI auto manages the widget with AUTO_WID_MNG. The keyword value specifies if the widget must have a defined value. Setting this keyword to “1” requires that the widget has either a default value or a user-entered value. Setting this keyword to “0” does not require a value. Do not use this keyword for user-managed widgets.

CEIL (optional)

Use this optional keyword to specify a maximum value allowed for input numbers. CEIL is only valid when the VALS keyword is specified.

DT (optional)

Use this optional keyword to specify the IDL data type of the edit values, using the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

FIELD (optional)

Use this optional keyword to specify the number of decimal places to use when formatting numbers. This keyword has no effect unless the data type is 4 (float) or greater. FIELD is only valid when the VALS keyword is specified.
FLOOR (optional)
Use this optional keyword to specify a minimum value allowed for input numbers. FLOOR is only valid when the VALS keyword is specified.

FRAME (optional)
Set this optional keyword to draw a frame around the widget.

LIST
Use this keyword to specify a string array of items to edit. If the VAL keyword is specified then LIST is the tag name associated with each value.

PROMPT (optional)
Use this optional keyword to specify the prompt string to be used for the widget.

SELECT_PROMPT (optional)
Use this optional keyword to specify a prompt string associated with the item being edited.

UVALUE
Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.

VALS (optional)
Use this optional keyword to provide an array of values to edit. The presence of this keyword requires that a string identifier be associated with each value.

XSIZE (optional)
Use this optional keyword to specify the width of the widget, in characters.

YSIZE (optional)
Use this optional keyword to specify the height of the widget, in characters.
Widget Event

When the widget is not auto managed, widget events set `event.result` to the edited list or, if VAL is specified, to the array of edited numbers. Input numbers are always returned as double precision regardless of the value of the DT keyword. Widget events occur any time an edit takes place.

Example

Create a simple compound widget for editing multiple values from a list. Print out the final edited list.

```plaintext
base = widget_auto_base(title='Edit test')
list = ['Item A', 'Item B', 'Item C', 'Item D']
vals = [10.0, 20.0, 30.0, 40.0]
we = widget_edit(base, uvalue='edit', list=list, $
vals=vals, /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'New Values', result.edit
```

See Also

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_PARAM
WIDGET_GEO

The WIDGET_GEO compound widget is used to specify Latitude and Longitude values. The function returns the base ID of the widget.

Calling Sequence

\[
\text{Result} = \text{WIDGET\_GEO(Base)}
\]

Arguments

Base

The widget ID of the widget base.

Keywords

AUTO\_MANAGE (optional)

Use this optional keyword to specify how ENVI auto manages the widget with AUTO\_WID\_MNG. The keyword value specifies if the widget must have a defined value. Setting this keyword to “1” requires that the widget has either a default value or a user-entered value. Setting this keyword to “0” does not require a value. Do not use this keyword for user-managed widgets.

DEFAULT (optional)

Set this optional keyword to specify the default Latitude and Longitude in decimal degrees. DEFAULT is a two element float array. If LAT\_ONLY is set then DEFAULT is a single element float array.

DMS (optional)

Set this optional keyword to specify that Latitude and Longitude in Degrees, Minutes and Seconds. Set DMS to zero to allow decimal degrees. The default is to use Degrees, Minutes and Seconds.

LAT\_ONLY

Set this keyword to specify that only Latitude values are shown.

PROMPT (optional)

Use this optional keyword to specify a string array for the prompt string. If no values are specified then the prompt array is set to [‘Lat’, ‘Lon’]
UVALUE

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.

Example

Create a simple compound widget to enter a Latitude and Longitude value. If the values are entered successfully then print out the result.

```plaintext
base = widget_auto_base(title='Simple Lat/Lon')
wg = widget_geo(base, /dms, uvalue='geo', /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'Latitude: ', result.geo(0)
print, 'Longitude: ', result.geo(1)
```

See Also

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_MAP
WIDGET_MAP

WIDGET_MAP is a specialized compound widget to allow input and edit of map coordinates and projections.

Calling Sequence

Result = WIDGET_MAP(Base)

Arguments

Base

The widget ID of the base widget.

Keywords

AUTO_MANAGE (optional)

Use this optional keyword to specify how ENVI auto manages the widget with AUTO_WID_MNG. The keyword value specifies if the widget must have a defined value. Setting this keyword to “1” requires that the widget has either a default value or a user-entered value. Setting this keyword to “0” does not require a value. Do not use this keyword for user-managed widgets.

DEFAULT_MAP (optional)

Use this optional keyword to set the default map location for the compound widget. DEFAULT_MAP is a double array with the following values:

- default_map(0) is the default X map location.
- default_map(1) is the default Y map location.

DEFAULT_PROJ (optional)

Use this optional keyword to set the default map projection for the compound widget. DEFAULT_PROJ is a projection structure returned from ENVI_GET_PROJECTION or ENVI_PROJ_CREATE.

PROJECTION (optional)

Use this optional keyword to specify a map projection. PROJECTION is a projection structure returned from ENVI_GET_PROJECTION or ENVI_PROJ_CREATE.
**FLIP (optional)**

Set this optional keyword to allow flipping between lat/long and map coordinates.

**FRAME (optional)**

Set this optional keyword to place a frame around the widget base.

**UVALUE**

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widget, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.

**Widget Event**

When the widget is not auto managed, widget events set `event.result` to a structure with three tags: “map_x”, “map_y”, and “proj”.

**See Also**

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_GEO
WIDGET_MENU

WIDGET_MENU is a specialized compound widget used to create a menu. The function returns the base ID of the widget.

Calling Sequence

\[
Result = \text{WIDGET\_MENU}(Base)
\]

Arguments

Base

The widget ID of the base widget.

Keywords

**AUTO\_MANAGE** (optional)

Use this optional keyword to specify how ENVI auto manages the widget with AUTO\_WID\_MNG. The keyword value specifies if the widget must have a defined value. Setting this keyword to “1” requires that the widget has either a default value or a user-entered value. Setting this keyword to “0” does not require a value. Do not use this keyword for user-managed widgets.

**BUT\_BASES** (optional)

Use this optional keyword to specify a named variable to hold the returned array of the widget ID of each button. Specifying this keyword allows you to manage each button by making calls to WIDGET\_CONTROL.

**DEFAULT\_ARRAY** (optional)

Use this optional keyword to specify an array indicating the state of each button. A one indicates the button is set, a zero indicates not set.

**DEFAULT\_PTR** (optional)

Use this optional keyword to specify which single button should be set by default.

**EXCLUSIVE** (optional)

Set this optional keyword to make the menu items exclusive.
LIST

Use this keyword to specify an array of string values for the menu buttons.

PROMPT (optional)

Use this optional keyword to specify the prompt string to be used for the widget.

ROWS (optional)

Use this optional keyword to specify the number of rows for the menu. The default is one.

UVALUE

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.

Widget Event

When the widget is not auto managed, if EXCLUSIVE is set, event.result is set to a single value indicating which button was pressed. If EXCLUSIVE is not set, event.result is a byte array of zeros and ones where a one indicates a selected button.

Example

Create a simple compound widget to select one item from an exclusive menu. If the menu item is properly selected then print out the result.

```idl
base = widget_auto_base(title='Menu test')
list = ['Button 1', 'Button 2', 'Button 3', 'Button 4']
wm = widget_menu(base, list=list, uvalue='menu', /excl, $ /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'Menu Selected', result.menu
```

See Also

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_PMENU
WIDGET_MULTI

WIDGET_MULTI is a compound widget that allows selection of multiple items from a list. The function returns the base ID of the widget. This is the widget used for spectral subsetting.

Calling Sequence

\[ \text{Result} = \text{WIDGET_MULTI}(\text{Base}) \]

Arguments

Base

The widget ID of the base widget.

Keywords

AUTO_MANAGE (optional)

Use this optional keyword to specify how ENVI auto manages the widget with AUTO_WID_MNG. The keyword value specifies if the widget must have a defined value. Setting this keyword to "1" requires that the widget has either a default value or a user-entered value. Setting this keyword to "0" does not require a value. Do not use this keyword for user-managed widgets.

BUTTON_BASSES (optional)

Use this optional keyword to specify a named variable to hold the returned array of the widget ID of each button. Specifying this keyword allows the user to manage each button by making calls to WIDGET_CONTROL.

DEFAULT (optional)

Use this optional keyword to specify a byte array of zeros and ones indicating the default selection state of each item in the multi list. A one indicates that the item is selected, a zero indicates it is not selected.

LIST

Use this keyword to specify an array of string values for the menu buttons.

NO_RANGE (optional)

Set this optional keyword to disable selection of items by specifying a range.
PROMPT (optional)
Use this optional keyword to specify the prompt string to be used for the widget.

SELECT_PROMPT (optional)
Use this optional keyword to specify a prompt string associated with the item being edited.

UVALUE
Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.

YSIZE (optional)
Use this optional keyword to specify the height of the widget in pixels. The default is 350.

Widget Event

When the widget is not auto managed, widget events set event.result to an array of zeros and ones, where a one indicates a selected item.

Example

Create a simple compound widget to select multiple items from a list. If the items are properly selected then print out the result.

```idl
base = widget_auto_base(title='Multi test')
list = ['Item 1', 'Item 2', 'Item 3', 'Item 4']
wm = widget_multi(base, list=list, uvalue='list', /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'Selected Items', where(result.list eq 1)
```

See Also

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_SLIST
WIDGET_OUTF

WIDGET_OUTF is an output file name selector that provides no option of output to memory. The function returns the base ID of the widget.

Calling Sequence

Result = WIDGET_OUTF(Base)

Arguments

Base
The widget ID of the base widget.

Keywords

AUTO_MANAGE (optional)
Use this optional keyword to specify how ENVI auto manages the widget with AUTO_WID_MNG. The keyword value specifies if the widget must have a defined value. Setting this keyword to “1” requires that the widget has either a default value or a user-entered value. Setting this keyword to “0” does not require a value. Do not use this keyword for user-managed widgets.

DIRECTORY (optional)
Set this optional keyword to allow selection of an output directory.

DEFAULT (optional)
Use this optional keyword to specify the default text string to place in the widget.

FUNC (optional)
Use this optional keyword to specify the name of a function to call with the input string. This allows you to call a custom routine to validate inputs. The function used in ENVI is ENVI_OUT_CHECK(), which returns a one if the entered filename is valid.

PROMPT (optional)
Use this optional keyword to specify the prompt string to be used for the widget.
**UVALUE**

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.

**XSIZE (optional)**

Use this optional keyword to specify the width of the widget, in characters.

**Widget Event**

When the widget is not auto managed, widget events set event.result to the input string.

**Example**

Create a simple compound widget to select a file. If the file is properly selected then print out the filename.

```plaintext
base = widget_auto_base(title='File Selection test')
wo = widget_outf(base, uvalue='outf', /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'Selected File', result.outf
```

**See Also**

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_OUTFM
WIDGET_OUTFM

WIDGET_OUTFM is an output file selector that includes the option of output to memory. The function returns the base ID of the widget.

Calling Sequence

\[ \text{Result} = \text{WIDGET_OUTFM}(\text{Base}) \]

Arguments

Base

The widget ID of the base widget.

Keywords

AUTO_MANAGE (optional)

Use this optional keyword to specify how ENVI auto manages the widget with AUTO_WID_MNG. The keyword value specifies if the widget must have a defined value. Setting this keyword to “1” requires that the widget has either a default value or a user-entered value. Setting this keyword to “0” does not require a value. Do not use this keyword for user-managed widgets.

DEFAULT (optional)

Use this optional keyword to specify the default text string to place in widget.

FRAME (optional)

Set this optional keyword to place a frame around the widget base.

FUNC (optional)

Use this optional keyword to specify the name of a function to call with the input string. This allows you to call a custom routine to validate inputs. The function used in ENVI is ENVI_OUT_CHECK(), which returns a one if the entered filename is valid and zero otherwise.

PROMPT (optional)

Use this optional keyword to specify the prompt string to be used for the widget.
**UVALUE**

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.

**XSIZE (optional)**

Use this optional keyword to specify the width of the widget, in characters.

**Widget Event**

When the widget is not auto managed, widget events set `event.result` to a structure with two tags: “name” and “in_memory”. If “in_memory” is zero then “name” is set to the input string. If “in_memory” is one then the user desires to leave the output in memory and not write the result to a file.

**Example**

Create a simple compound widget to select a file/memory. If the file/memory is properly selected then print out the result.

```plaintext
base = widget_auto_base(title='File/Memory Selection test')
wo = widget_outfm(base, uvalue='outf', /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
if (result.outf.in_memory) eq 1) then $print, 'Output to memory selected'$
else $print, 'Selected File', result.outf.name
```

**See Also**

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_OUTF
WIDGET_PARAM

WIDGET_PARAM is a compound widget used to enter a parameter. The function returns the base ID of the widget.

Calling Sequence

Result = WIDGET_PARAM(Base)

Arguments

Base
The widget ID of the base widget.

Keywords

AUTO_MANAGE (optional)
Use this optional keyword to specify how ENVI auto manages the widget with AUTO_WID_MNG. The keyword value specifies if the widget must have a defined value. Setting this keyword to “1” requires that the widget has either a default value or a user-entered value. Setting this keyword to “0” does not require a value. Do not use this keyword for user-managed widgets.

CEIL (optional)
Use this optional keyword to specify a maximum value allowed for input numbers.

COMMA (optional)
Set this optional keyword to format numbers with commas (i.e. 3,124,000).

CM (optional)
Set this optional keyword to cause the entered parameter to be annotated with “cm.” CM cannot be used with INCHES or PERCENT.

DEFAULT (optional)
Use this optional keyword to set the default value for the parameter.
**DT (optional)**

Use this optional keyword to specify a named variable that will contain the IDL data type of the file, using the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.

**FIELD (optional)**

Use this optional keyword to specify the number of decimal places to use when formatting numbers. This keyword has no effect unless the data type is 4 (float) or greater.

**FLOOR (optional)**

Use this optional keyword to specify a minimum value allowed for input numbers.

**INCHES (optional)**

Set this optional keyword to cause the entered parameter to be annotated with “” (the inch symbol). INCHES cannot be used with CM or PERCENT.

**PERCENT (optional)**

Set this optional keyword to cause the entered parameter to be annotated with “%” (the percent symbol). PERCENT cannot be used with CM or INCHES.

**PROMPT (optional)**

Use this optional keyword to specify the prompt string to be used for the widget.

**UNDEFINED (optional)**

Use this optional keyword to specify the value returned when the parameter is undefined. The default returned value is $10^{-34}$.

**UVALUE**

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.
**XSIZE (optional)**

Use this optional keyword to specify the width of the widget, in characters.

**Widget Event**

When the widget is not auto managed, widget events set `event.result` to the value of the input parameter. If the parameter is undefined then the value of the keyword UNDEFINED is returned. Note: The data type of `event.result` is always double. The user is responsible for casting the return value to the appropriate data type.

**Example**

Create a simple compound widget for inputting one floating point parameter. If the parameter is properly entered then print the result.

```python
base = widget_auto_base(title='Parameter test')
we = widget_param(base, dt=4, field=3, floor=0., default=10., uvalue='param', /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'Parameter value = ', float(result.param)
```

**See Also**

`AUTO_WID_MNG`, `WIDGET_AUTO_BASE`, `WIDGET_EDIT`
WIDGET_PMENU

WIDGET_PMENU is a pulldown menu compound widget. The function returns the base ID of the widget.

Calling Sequence

\[ \text{Result} = \text{WIDGET_PMENU}(\text{Base}) \]

Arguments

Base

The widget ID of the base widget.

Keywords

AUTO_MANAGE (optional)

Set this optional keyword to specify that the widget be managed by the ENVI function AUTO_WID_MNG.

BUT_BASES (optional)

Use this optional keyword to specify a named variable to hold the returned array of the widget ID of each button. Specifying this keyword allows the user to manage each button by making calls to WIDGET_CONTROL.

DEFAULT (optional)

Use this optional keyword to specify the index entry of the default menu selection.

LIST

Use this keyword to specify a string array of pulldown menu items. For the best results, pad list items with spaces to make all items equal in length.

LOOKUP (optional)

Use this optional keyword to specify an array of values associated with each menu item.

PROMPT (optional)

Use this optional keyword to specify the prompt string to be used for the widget.
UVALUE

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.

XSIZE (optional)

Use this optional keyword to specify the width of the widget, in characters.

Widget Event

When the widget is not auto managed, widget events set event.result to the index of the selected menu item.

Example

Create a simple compound widget for a pulldown menu. If OK is selected then print the menu item selected.

```python
base = widget_auto_base(title='Menu test')
list = ['Item 1', 'Item 2', 'Item 3', 'Item 4']
we = widget_pmenu(base, list=list, uvalue='menu', /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'Menu Item Selected= ', list(result.menu)
```

See Also

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_MENU
WIDGET_RGB

WIDGET_RGB is a compound widget to modify a color value associated with an rgb using the RGB, HLS, or HSV color system. The function returns the base ID of the widget.

Calling Sequence

Result = WIDGET_RGB(Base)

Arguments

Base
The widget ID of the base widget.

Keywords

AUTO_MANAGE (optional)
Set this optional keyword to specify that the widget be managed by the ENVI function AUTO_WID_MNG.

BIT_24 (optional)
Set this optional keyword to specify 24-bit color.

INDEX (optional)
Use this optional keyword to select the color table index to edit. The default is zero.

UVALUE
Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.
**Widget Event**

When the widget is not auto managed, widget events set `event.result` to a byte array of three elements, RGB, regardless of the color system. User managed widgets can use `WIDGET_CONTROL SET_VALUE` to change the color index being edited. In this case, `SET_VALUE` should be set to a 3 element array of RGB values for the current index. Optionally `SET_VALUE` can be set to a 4 element array where the fourth value is a new color index to edit.

`WIDGET_CONTROL GET_VALUE` can be used to return the current RGB 3-element array. The inputs and output from `SET_VALUE` and `GET_VALUE` are always RGB regardless of the color system selected.

**Example**

Create a simple compound widget to modify the fifth color in the ENVI graphic color table. If the color is properly modified then output the new RGB color triplet.

```python
base = widget_auto_base(title='RGB test')
we = widget_rgb(base, index=5, uvalue='rgb', /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'New RGB color triplet= ', result.rgb
```

**See Also**

`AUTO_WID_MNG`, `WIDGET_AUTO_BASE`, `ENVI_GET_RGB_TRIPLETS`
WIDGET_SLABEL

WIDGET_SLABEL is a compound widget used to display a string message with scroll bars. The function returns the base ID of the widget.

Calling Sequence

\[ \text{Result} = \text{WIDGET_SLABEL}(\text{Base}) \]

Arguments

Base
The widget ID of the base widget.

Keywords

FRAME (optional)
Set this optional keyword to create a frame around the widget.

PROMPT (optional)
Use this optional keyword to specify the text to display.

XSIZE (optional)
Use this optional keyword to specify the width of the widget, in characters.

YSIZE (optional)
Use this optional keyword to specify the height of the widget, in characters.

Widget Event

This is a passive widget and does not generate events.

See Also

WIDGET_AUTO_BASE
WIDGET_SLIST

WIDGET_SLIST is a compound widget for creating lists. The function returns the base ID of the widget.

Calling Sequence

\[ \text{Result} = \text{WIDGET_SLIST}(\text{Base}) \]

Arguments

Base

The widget ID of the base widget.

Keywords

AUTO_MANAGE (optional)

Use this optional keyword to specify how ENVI auto manages the widget with AUTO_WID_MNG. The keyword value specifies if the widget must have a defined value. Setting this keyword to “1” requires that the widget has either a default value or a user-entered value. Setting this keyword to “0” does not require a value. Do not use this keyword for user-managed widgets.

DEFAULT (optional)

Use this optional keyword to specify the list item that is selected by default.

FRAME (optional)

Set this optional keyword to create a frame around the compound widget.

LIST

Use this keyword to specify a string array of items in the selection list.

NO_SELECT (optional)

Normally, the item selected from the list is displayed in a separate text box. Set this optional keyword to prevent the selected item from being displayed outside the list.

PROMPT (optional)

Use this optional keyword to specify the prompt string to be used for the widget.
SELECT_PROMPT (optional)

Use this optional keyword to specify a string to be used as the prompt appearing in the selection widget.

UVALUE

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.

XSIZE (optional)

Use this optional keyword to specify the width of the widget, in characters.

YSIZE (optional)

Use this optional keyword to specify the height of the widget, in characters.

Widget Event

When the widget is not auto managed, widget events set event.result to the index of the selected item.

See Also

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_MULTI
WIDGET_SSLIDER

WIDGET_SSLIDER is a compound widget for setting values using a slider. The function returns the base ID of the widget.

Calling Sequence

Result = WIDGET_SSLIDER(Base)

Arguments

Base

The widget ID of the base widget.

Keywords

AUTO_MANAGE (optional)

Set this optional keyword to specify that the widget be managed by the ENVI function AUTO_WID_MNG.

CEIL (optional)

Use this optional keyword to specify a maximum value allowed for input numbers. The returned value will always equal to or less than this number, even if the slider is positioned higher.

DRAG (optional)

Set this optional keyword to cause events to be generated continuously while the slider is being moved. For more information see the DRAG keyword to the IDL WIDGET_SLIDER function.

DT (optional)

Use this optional keyword to specify the data type of the slider values, using the following IDL convention: 1=byte (8-bits), 2=integer (16-bits), 3=long integer (32-bits), 4=floating-point (32-bits), 5=double-precision floating point (64-bits), 6=complex (2x32-bits), 9=double-precision complex (2x64-bits), 12=unsigned integer (16-bits), 13=unsigned long integer (32-bits), 14=long 64-bit integer, 15=unsigned long 64-bit integer.
FIELD (optional)

Use this optional keyword to set the number of decimal places used to format numbers. This keyword has no effect unless the data type is 4 (float) or greater.

FLOOR (optional)

Use this optional keyword to set the minimum value allowed for input numbers. The returned value will always be equal to or greater than this number, even if the slider is positioned lower.

INCHES (optional)

Set this optional keyword to cause the slider to be annotated with “” (the inch symbol).

MAX

Use this keyword to specify an integer maximum for the slider. The value of MAX will be multiplied by the value of SCALE to arrive at the actual maximum slider value.

MIN

Use this keyword to specify an integer minimum for the slider. The value of MIN will be multiplied by the value of SCALE to arrive at the actual minimum slider value.

SCALE

Use this keyword to specify a multiplicative scale factor to be used in conjunction with the MIN and MAX keywords. The default is 1.

TITLE

Use this keyword to specify a string that will appear as the title of the slider.

UVALUE

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.
VALUE

Use this keyword to set an initial slider value. The SCALE factor is not applied.

XS (optional)

Use this optional keyword to provide a two-element array. The first element is the slider size in pixels, and the second element is the text input widget size in characters. The default array is [150, 7].

Widget Event

When the widget is not auto managed, widget events set event.result to the slider value. This value is already scaled by the SCALE factor. The returned value is always of type double and should be cast to the user’s selected data type.

Example

Create a simple compound widget for selection a value using a slider. Print out the final slider location.

```plaintext
base = widget_auto_base(title='Simple Slider')
wg = widget_sslider(base, title='Samples', min=0, max=10, $value=0, uvalue='slide', /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'Slider value', result.slide
```

See Also

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_PARAM
WIDGET_STRING

WIDGET_STRING is a compound widget used to enter a string. The function returns the base ID of the widget.

Calling Sequence

Result = WIDGET_STRING(Base)

Arguments

Base

The widget ID of the base widget.

Keywords

ALL_EVENTS (optional)

Set this optional keyword to specify that the widget generate an event for all supported events.

AUTO_MANAGE (optional)

Set this optional keyword to specify that the widget be managed by the ENVI function AUTO_WID_MNG.

DEFAULT (optional)

Use this optional keyword to specify a default string.

PROMPT (optional)

Use this optional keyword to specify the prompt string to be used for the widget.

UVALUE

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.
XSIZE (optional)

Use this optional keyword to specify the width of the widget, in characters.

Widget Event

When the widget is not auto managed, widget events set event.result to the entered string.

Example

Create a simple compound widget for selection a value using a slider. Print out the final slider location.

```python
base = widget_auto_base(title='Edit String')
ws = widget_string(base, uvalue='str', /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'New String ', result.str
```

See Also

AUTO_WID_MNG, WIDGET_AUTO_BASE, WIDGET_PARAM
WIDGET_SUBSET

WIDGET_SUBSET is a compound widget used to specify image subsets. The function returns the base ID of the widget.

Calling Sequence

\[ \text{Result} = \text{WIDGET_SUBSET}(\text{Base}) \]

Arguments

Base

The widget ID of the base widget.

Keywords

AUTO_MANAGE (optional)

Set this optional keyword to specify that the widget be managed by the ENVI function AUTO_WID_MNG.

DIMS

Use this keyword to specify a named variable that contains a five-element array holding the initial data dimensions. The elements are defined as follows:

- DIMS(0): a pointer to the ROI (set to -1 if no ROI is selected).
- DIMS(1): the starting X coordinate.
- DIMS(2): the ending X coordinate.
- DIMS(3): the starting Y coordinate.
- DIMS(4): the ending Y coordinate.

FID

Use this keyword to specify the file ID of the file in use. The file ID is used to get the file name and the number of samples and lines.

ROI (optional)

Set this optional keyword to allow ROI subsetting of a file. The ROI ID is returned as the first elements in the DIMS array.
UVALUE

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.

XS (optional)

Use this optional keyword to specify the X size in characters

Widget Event

When the widget is not auto managed, widget events set event.result to the DIMS array.

Example

Create a simple compound widget for selecting image subsets. Print out the new dims array.

```python
envi_file_query, fid, ns=ns, nl=nl
dims = [-1, 0, ns-1, 0, nl-1]
base = widget_auto_base(title='Subset test')
ws = widget_subset(base, uvalue='subset', fid=fid, 
                  dims=dims, /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'New dims', result.subset
```

See Also

AUTO_WID_MNG, WIDGET_AUTO_BASE, ENVI_SELECT
WIDGET_TOGGLE

The WIDGET_TOGGLE compound widget is used to create arrow toggle selections. The function returns the base ID of the widget.

Calling Sequence

\[ \text{Result} = \text{WIDGET_TOGGLE}(\text{Base}) \]

Arguments

Base

The widget ID of the widget base.

Keywords

AUTO_MANAGE (optional)

Set this optional keyword to specify that the widget be managed by the ENVI function AUTO_WID_MNG.

DEFAULT (optional)

Use this optional keyword to specify the list item that is selected by default.

LIST

Use this keyword to specify a string array of items in the selection list.

PROMPT (optional)

Use this optional keyword to specify the prompt string to be used for the widget.

UVALUE

Use this keyword to assign a “user value” to the widget. This value may be of any data type and organization. The user value exists entirely for the convenience of the programmer. The UVALUE can be retrieved by using WIDGET_CONTROL.

For widgets managed by the ENVI function AUTO_WID_MNG, UVALUE is a tag name in the returned anonymous structure. For user-managed widgets, UVALUE can be set and used as desired. UVALUE must be set for all compound widgets.
XSIZE (optional)

Use this optional keyword to specify the width of the widget, in characters.

Example

Create a simple compound widget for displaying a toggle selection. Print out the final toggle selection.

```python
base = widget_auto_base(title='Toggle test')
list = ['Item A', 'Item B', 'Item C']
sb = widget_base(base, /row)
wt = widget_toggle(sb, uvalue='toggle', list=list, /auto)
result = auto_wid_mng(base)
if (result.accept eq 0) then return
print, 'Toggle Selected', result.toggle
```
Index

Numerics
3-D Image Cube Routine, 370
3-D Image Cube User Function, 370

A
adapt_filt_doit, 217, 217
Adaptive Filtering, 217, 217
AIRSAR
   airsar_header program, 221
   airsar_scatter_doit program, 232
Header Information, 221
Pedestal Height, 223
Phase Image, 226
Polarization Signatures, 229
Scatter Classification, 232
Synthesize Image, 236

airsar_header_doit, 221
airsar_ped_height_doit, 223
airsar_phase_image_doit, 226
airsar_polsig_doit, 229
airsar_scatter_doit, 232
airsar_synth_doit, 236
Annotation
   User-Defined, 536
arguments, described, 203
Aspect Corrections, 240
aspect_doit, 240, 240
auto_wid_mng, 243
Auto-Managed Widget Events, 92
AVHRR
   Calibrate Data, 328
   Compute Geometry, 331
   Warp Data, 334
B

Bad Lines Correction, 244
bad_data_doit, 244, 244
Band Math
  Developing Functions, 43
  Example Function, 45
  Introduction, 28
  math_doit program, 576
  Program, 576
Band Ratios, 616, 616
Band Selection, 627
Batch Execution
  Available Routines, 205
  Examples, 65, 66
  Program Specifications
    Apply Lookup Table (LUT), 562
  Programming, 65
  Recording, 59
Batch Mode
  ENVI Message Logging, 61
  Exiting, 56, 339
  Helpful Tips, 62
  Hybrid, 53
  Initializing, 341
  Initiating, 55
  Log File, 61
  Making a Shortcut, 62
  Running, 53
  Using envi_open_data_file, 37
  Writing Programs, 57
Batch Processing, 52
Buffer Zone Image, 345
Build FFT Filter, 420

C

Calibrate, 348
Calibrate AVHRR Data, 243
Calibrate TIMS, 701
Calibration, 348

Calling Sequence
  Functions, 202
  calling sequence
    procedures, 202
CATCH, 100
  catch, 247
cf_doit, 248, 248
Class Statistics, 276
class_confusion_doit, 251
class_cs_doit, 256
class_doit, 260
class_majority_doit, 270
class_rule_doit, 273
class_stat_doit, 276
Classification
  class_confusion_doit program, 251
class_cs_doit program, 256
class_doit program, 260
class_rule_doit program, 273
class_stats_doit program, 276
com_class_doit program, 280
Create Buffer Zone Image., 345
Create Segmentation Image, 497
evi_clover_doit program, 352
Majority/Minority Analysis, 270
Neural Net, 468
Supervised, 260
Classify Rule Images, 273
Closing Files, 131
Clump and Sieve, 256
Color Transforms
  Forward, 632
  Inverse, 629
  rgb_itrans_doit program, 629
  rgb_trans_doit program, 632
com_class_doit, 280
Combine Classes, 280
COMPILE_OPT, 120
Compiling Code, 30
Compound Widgets
evi_info_wid, 451
widget_map, 740
Compound Widgets Offsets, 351
Compute Confusion Matrix, 251
Compute Sun Angles, 358
Configuration Values, 422
Continuum Removal, 283
continuum_remove_doit, 283
Contrast Stretch, 687
cov_doit, 286, 286
Conversions, 289
covert_doit, 289, 289
covert inplace_doit, 291
Converting
   File Coordinates, 360
   Interleave Types, 289
   Interleave Types in Place, 291
   Map Projection, 362
   Projection Coordinates, 366
Convolution Filtering, 286, 286
Coordinates
   Converting, 360
   File, 168
   Image, 168
   Pixel, 168
   Systems in ENVI, 168
Crashes, 79
Create New File, 248
Creating Reports, 127
cross_track_correction_doit, 293
Cross-Track Illumination, 293
cube_3d_doit, 370

dem_bad_data_doit, 303
Deskew MSS, 306
deskew_doit, 306
deskew_doit program, 306
Destripe data, 309
Destripe Image Data, 309
destripe doit, 309
destripe doit program, 309
DIMS, 34
DIMS Pointer, 178
disp_get_location, 312
disp_goto, 314
disp_out_img, 317
disp_out_img function, 317
Display Grayscale or RGB, 383
Display Information, 428
Display Numbers List, 426

E
Edit Multiple Values, 735
eline_cal doit, 320
eline_cal doit program, 320
Emissivity Conversion, 323
emittance_calc_doit, 323
Empirical Line Calibration, 320, 320
Endmember Collection, 355
   Widget, 182
Enter Image Data, 389
ENVI
   Coordinate Systems in, 168
   File Input/Output, 32
   and IDL, 18
   Installation Directory Components, 187
   Library Directories, 33
   Library Functions, 196
   Library Routines Reference, 190
   List of Routines, 205
   Managing Files, 36
   Routine Name, 326
   Variable Codes, 196
Index

envis, 326
ENVI File
  Creating New from Existing, 40
  Format, 39
ENVI Projection
  Structure, 483
  Update, 327
ENVI Runtime
  Adapting User Functions for, 120
  Version, 18, 28
ENVI Vector File
  Add Record, 398
  Close, 396, 400
  DBF Attribute File, 543
  Get Information, 404
  New, 402
  Open, 406
  Retrieve Records, 408
  envi_add_projection, 327
  envi_avhrr_calibrate_doit, 328
  envi_avhrr_geometry_doit, 331
  envi_avhrr_warp_doit, 334
  envi_batch_exit, 339
  envi_batch_init, 341
  envi_batch_status_window, 342
  envi_buffer_zone_doit, 345
  envi_cal_doit, 348, 348
  envi_center, 351
  envi_clover_doit, 352
  envi_collect_spectra, 355
  envi_compute_sun_angles, 358
  envi_convert_file_coordinates, 360
  envi_convert_file_map_projection, 362
  envi_convert_projection_coordinates, 366
  envi_create_roi, 369
  envi_cube_3d_doit, 370
  envi_default_stretch_create, 373
  envi_define_roi, 376
  envi_delet_rois, 379
  envi_disp_query, 380
  envi_display_bands, 383
  envi_doit, 386
  envi_enter_data, 389
  envi_evf_close, 396
  envi_evf_define_add_record, 398
  envi_evf_define_close, 400
  envi_evf_define_init, 402
  envi_evf_info, 404
  envi_evf_open, 406
  envi_evf_read_record, 408
  envi_file_mng, 410
  envi_file_query, 411
  envi_file_type, 417
  envi_filter_doit, 420
  envi_get_configuration_values, 422
  envi_get_data, 423, 423
  envi_get_display_numbers, 426
  envi_get_file_ids, 427
  envi_get_image, 428
  envi_get_map_info, 430
  envi_get_projection, 431
  envi_get_roi, 435
  envi_get_roi_data, 436
  envi_get_roi_dims_ptr, 438
  envi_get_roi_ids, 439
  envi_get_slice, 441
  envi_get_statistics, 443
  envi_get_tile, 446
  envi_grid_doit, 448, 448
  envi_info_wid, 451
  envi_init_tile, 452
  envi_io_error, 454
  envi_layer_stacking_doit, 455
  envi_map_info_create, 460
  envi_mask_apply_doit, 465, 465
  envi_neural_net_doit, 468
  envi_open_data_file, 473
  envi_open_file, 476
  envi_output_to_external_format, 477
  envi_pickfile, 479
  envi_plot_data, 481
  envi_proj_create, 483
environ

ASCII Column Data, 487
environ_report_inc, 488
environ_report_init, 489
environ_report_stat, 491
environ_restore_roi, 493
environ_save_rois, 496
environ_segment_doit, 497
environ_select, 500
environ_sensor_type, 504
environ_set_inheritance, 507
environ_setup_head, 510
environ_stats_doit, 519
environ_sum_data_doit, 525
environ_synthetic_color_doit, 529
environ_tile_done, 532
environ_toggle_catch, 533
environ_translate_projection_name, 534
environ_translate_projection_units, 535
environ_user_defined_annotation, 536
environ_write_dbf_file, 543
environ_write_statistics, 545

Error
  Catching, 31
  Checking, 80
  Handling, 98
Input/Output, 605
  Trapping in User Functions, 98
  Using CATCH, 100
error catching, 533

Examples
  Band Math Function, 45
  Basic Image Information, 129
  Batch File Statistics, 65
  Batch Saturation Stretch, 66
  Calculating ROI Means, 177
  Choosing Files Interactively, 132
  Creating a Report, 127
  Getting RGB Color Values, 128
  I/O Error Handling, 99
  Map Information, 130

Non-tiled Spatial Processing, 115
Non-tiled Spectral Processing, 116
Parsing Image Headers, 155, 156
Plot Function, 137
Plotting, 126
Processing Status Dialog, 118
Restore Saved ROIs, 175
ROI Addresses, 180
ROI DIMS Pointer, 178
ROI Selection, 173
ROI Selection and WIDGET_MULTI, 174
Saving Spatial Tiles to Disk, 110
Saving Spatial Tiles to Memory, 112
Saving Spectral Tiles to Disk, 113
Spatial and Spectral Tiling, 107
Spatial Read Routine, 161
Spatial Tiling Example, 104
Spectral Analyst Function, 140
Spectral Math, 48
Spectral Read Routine, 164
Spectral Tiling, 105
User Define Move Routine, 150
User Defined Move Routine, 149
Using ROI Data, 176

Exceptions and Errors, 247

F

FFT
  Forward Transform, 548
  Inverse Transform, 551
FFT Filtering
  envi_filter_doit program, 420
  fft_doit program, 548
  fft_inv_doit program, 551
  Filter Definition, 420
  fft_doit, 548
  fft_inv_doit, 551
FID
  Defined, 34
  Set to -1, 34
Index

ENVI Programmer's Guide

File
  Choosing Filenames, 479
  Coordinates, 168
  Extracting Information, 129
  Getting File Type Description, 417
  ID, 427
  Management, 131
  Selection and Subsetting, 500
File Input
  Basics, 154
  Custom Read Routines, 158
  Parsing Headers, 155
  Spatial Read Routine, 160
  Spectral Read Routine, 163
File Management, 410
Filenames, 479
Filtering
  Adaptive, 217
  Convolutions, 286
  Match, 568
  Mixture Tuned Match Filtering, 572
  Morphological, 585
formal parameters, 203
FORWARD_FUNCTION, 120
Functions Calling Sequence, 202

G

Gain and Offset User Function, 554
gainoff_doit, 554
gen_image_doit, 557
General Information
  ENVI and IDL, 18
  Hardware platforms, 16
  Introduction to ENVI, 16
  Relationship of ENVI to IDL, 18
  Review of ENVI Functionality, 19
Georeferenced Image
  Building Multiband File from, 455
  Displaying Map Information, 430
  Mosaicking, 588
Gridded Points to Raster, 448, 448

H

Handle Value, 560
handle_value, 560, 560
Handles, 560
Hardware platforms, 16
Header File Information, 510
hist_export_doit, 562
Histograms
  hist_export_doit program, 562
Hybrid Batch Mode, 53

I

IDL
  and ENVI, 18
  File Input/Output, 32
  Library Directories, 33
  Multiple Versions of, 55
  Using ENVI Routines, 57
Image
  Accessing Image Data, 38
  Controlling Displays, 185
  Coordinates, 168
  Displays, 101
  Georeferencing, 460
  Making ENVI Format Files, 39
  Saving Data to Disk, 39
  Saving Data to Memory, 39
  Inheritance Structure, 507
Input
  Error Handling, 98, 605
Installation Components, 187
Interactive
  Plot Function, 135
  Spectral Analyst, 138
  User Defined Map Projection Types, 142
  User Defined Move Routine, 148
User Defined Units, 147
Introduction to ENVI, 16
Inverse MNF Transform, 583

K
Keywords
Setting
  Setting Keywords, 203
keywords
described, 203
meaning of slash character, 203

L
Library Functions, 196
Library Routines, 190
Log File in Batch Mode, 61

M
M_FID, 34
magic_mem_check, 566
Majority Analysis, 270
Managing Files, 131
Map Information
  from Display, 430
  Example, 130
Map Projection
  Converting, 362
  Creating, 460
map_proj Directory, 189
Masking, 465, 465
Match Filtering, 568, 568, 572
match_filter_doit, 568
match_filter_mt_doit, 572
Matched Filtering
  match_filter_mt_doit program, 572
matched_filter_doit, 568
math_doit, 576
Memory Cache Clear, 566
Menu
  Directory, 188
  Item User Values, 77
  Modify ENVI Menus, 75
  Working with Menu Files, 75
Message Logging, 61
Minimum Noise Fraction (MNF), 579
Minority Analysis, 270
Mixture Tuned Match Filtering, 572
MNF Transform, 579
mnf_doit, 579
mnf_inv_doit, 583
morph_doit, 585, 585
Morphological Filtering, 585
mosaic_doit program, 588
Mosaicking Images, 588
MSS Aspect ratio, 240
MSS Calibration, 594
MSS Deskew, 306
msscal_doit, 594
Multilook SIR-C Data, 657
Munsell to RGB, 598
Munsell Transform, 598, 600
munsell_doit, 598
munsell_inv_doit, 600

N
named variables, 203, 204
NDVI (Normalized Difference Vegetation Index), 602
ndvi_doit, 602, 602
Neural Net Classification, 468
Non-tiled Processing, 115
nvi_get_rgb_triplets, 433

O
on_ioerror, 605
Open Data File, 473
Open ENVI File, 476
Opening Files, 131
Output
  Creating file from Bands in ABL, 248
Error Handling, 98, 605
to External Format, 477
  Image Using Lookup Table, 562
Overlay Classes, 352

P
Parsing Image Headers, 155
pc_rotate, 606
Pedestal Height Image
  airsar_ped_height_doit program, 223
  sirc_ped_height_doit program, 661
Phase Image
  airsar_phase_image_doit program, 226
  sirc_phase_image_doit program, 665
Pixel
  Coordinates, 168
  Display Location, 312
  Move Location, 314
Pixel Purity Index (PPI), 610
Plot Display, 481
Plot Function, 135
Plotting, 125
Pointers and Handles, 560
Polarization Signatures
  airsar_polsig_doit program, 229
  sirc_polsig_doit program, 669
POS, 35
positional parameters, 203
Postscript Output, 317
ppi_doit, 610
Principal Components
  pc_rotate, 606
Principal Components Rotation, 606
procedures calling sequence, 202
Processing Routines See. Routines
Processing Status Report, 117, 489
Processing Status Update, 491
Program Names
  adapt_filt_doit, 217
  airsar_header, 221
  airsar_header_doit, 221
  airsar_ped_height_doit, 223
  airsar_phase_image_doit, 226
  airsar_polsig_doit, 229
  airsar_scatter_doit, 232
  airsar_synth_doit, 236
  aspect_doit, 440
  bad_data_doit, 244
  cf_doit, 248
  class_confusion_doit, 251
  class_cs_doit, 256
  class_doit, 260
  class_rule_doit, 273
  class_stats_doit, 276
  com_class_doit, 280
  continuum_remove_doit, 283
  conv_doit, 286
  convert_doit, 289
  cross_track_correction_doit, 293
  cube_3d_doit, 370
  dark_sub_doit, 297
  decor_doit, 300
  deskew_doit, 306
  destripe_doit, 309
  disp_get_location, 312
  disp_goto, 314
  disp_out_img, 317
  eline_cal_doit, 320
  envi, 326
  envi_batch_exit, 339
  envi_batch_init, 341
  envi_batch_status_window, 342
  envi_cal_doit, 348
  envi_clover_doit, 352
  envi_default_stretch_create, 373
  envi_doit, 386
envi_filter_doit, 420
envi_get_file, 427
envi_grid_doit, 448
envi_layer_stacking_doit, 455
envi_mask_apply_doit, 465
envi_set_inheritance, 507
envi_stats_doit, 519
fft_doit, 548
fft_inv_doit, 551
gainoff_doit, 554
gen_image_doit, 557
hist_export_doit, 562
match_filter_doit, 568
match_filter_mt_doit, 572
math_doit, 576, 576
mnf_doit, 579
morph_doit, 585
mosaic_doit, 588
msscal_doit, 594
munsell_doit, 598
munsell_inv_doit, 600
ndvi_doit, 602
pc_rotate, 606
ppl_doit, 610
radar_inc_angle_doit, 613
ratio_doit, 616, 616
reg_warp_doit, 619
resize_doit, 624
rgb_itran_doit, 629
rgb_trans_doit, 632
roi_thresh_doit, 638
rotate_doit, 641
rtv_doit, 644
sat_stretch_doit, 647
sharpen_doit, 650
sirc_header_doit, 654
sirc_multilook_doit, 657
sirc_ped_height_doit, 661
sirc_phase_image_doit, 665
sirc_polsig_doit, 669, 673
slice_doit, 678
slt2gnd_doit, 680
spectral_feature_doit, 683
stretch_doit, 687
tascap_doit, 691
texture_cooccur_doit, 694
texture_stats_doit, 698
tmcal_doit, 704
topo_doit, 708
unmix_doit, 717
vax_ieee_doit, 720
Programming in ENVI
Band Math, 43
Batch
Examples, 65, 66
Programming, 65
Recording, 59
Compiling, 30
ENVI Menu Files, 30
ENVI Routines, 205
File Information, 129
File Input
Basics, 154
Custom Read Routines, 158
Parsing Headers, 155
Spatial Read Routine, 160
Spectral Read Routine, 163
File Management, 131
Interactive
Plot Function, 135
Spectral Analyst, 138
User Defined Map Projection Types, 142
User Defined Move Routine, 148
User Defined Units, 147
Introduction, 32
Plotting, 125
Processing Routines, 29

ENVI Programmer's Guide Index
Processing Status Report, 117
Region of Interest
  Addresses, 180
  Basics, 171
  Dims Pointer, 178
  Getting Data, 176
  Reports, 127
  RGB Triplets, 128
  Saving Processing Results, 109
  Spectral Math Functions, 47
  Toggle Catch, 31
  What is Possible, 28
  Projection Information, 431

Q
  Query Display, 380
  Query Header File, 411

R
  R_FID, 34
  Radar Incidence Angle Image, 613
  radar_inc_angle_doit, 613
  Raster To Vector, 644
  Raster to Vector
    rtv_doit program, 644
  ratio_doit, 616, 616
  Read Routines, 158
  Receiver Operating Characteristic Curves, 635
  Recording Batch, 59
  reg_warp_doit, 619
  Region of Interest
    Addresses, 180
    Creating, 369
    Defining, 376
    Deleting, 379
    Dims Pointer, 178
    Dims Pointer Value, 438
    Example, 173, 174, 175, 176, 177, 178, 180
  Getting Associated Data, 436
  Getting Data, 176
  Getting ID’s, 439
  Getting Point Addresses, 435
  Processing, 171
  Restoring Saved, 493
  Saving, 496
  Selecting, 172
  Threshold, 638
  Relationship of ENVI to IDL, 18
  Remove Bad Data, 244
  Report Input/Output Processing Errors, 454
  Report Widget, 451
  Reports, 127
  Resize images, 624
  resize_doit, 624
  RESOLVE_ALL, 120
  Retrieve Data from ABL, 423
  RGB Colors, 128
  RGB to Munsell, 600
  RGB Triplets, 433
  rgb_get_bands, 627
  rgb_itrans_doit, 629
  rgb_trans_doit, 632
  ROC Curves, 635
  roc_curve_doit, 635
  ROI See. Region of Interest
  roi_thresh_doit, 638
  Rotate Images, 641
  Rotate/Flip Data, 641
  rotate_doit, 641
  Routines
    Common Keywords, 34
    Compiling Dependent, 120
    Controlling without Interaction, 32
    Interface for Processing Routines, 386
    List of Available, 205
    Non-tiled Processing, 115
    Processing Routines, 102
    Tiled Processing, 102
    Using ENVI in IDL Programs, 57
rtv_doit, 644

S
sat_stretch_doit, 647
Saturation Stretch, 647
Save File
Making, 121
Saving
Image Data to Disk, 39
Image Data to Memory, 39
Processing Results, 109
Segmentation, 497
Sensor Type Values, 504
sharpen_doit, 650
Sharpening, 650
sharpen_doit, 650
SIRC
Header Information, 654
Multilook, 657
Pedestal Height, 661
Phase Image, 665
Polarization Signatures, 669
Synthesize Images, 673
sirc_header_doit, 654
sirc_multilook_doit, 657
sirc_multilook_doit program, 657
sirc_ped_height_doit, 661
sirc_phase_image_doit, 665
sirc_polsig_doit, 669, 673
Slant to Ground Range, 680
slash character, 203
slice_doit, 678
slt2gnd_doit, 680
Spatial Read Routine, 158, 160
Spatial Tiling Example, 107
Spectral Analyst Functions, 138
Spectral Feature Fitting, 683
Spectral Math
Developing Functions, 47
Example Function, 48
Introduction, 28
Spectral Read Routine, 158, 163
Spectral Slice, 441, 678
Spectral Tiling Example, 105, 107
Spectral Unmixing, 717
spectral_feature_doit, 683
spectral_feature_doit program, 683
Statistics
Calculate Parameters, 525
Calculating, 519
Reading Data, 443
Writing to File, 545
Status
Displaying the Window, 342
Report, 117
Stretch
Contrast (File-to-File), 687
Default Structure, 373
stretch_doit, 687
STRICTARR, 120
Sum Data Bands, 525
Supervised Classification, 260
Synthesize JPL AIRSAR Data, 236
Synthesize SIR-C Images, 673
Synthetic Color Image, 529
T
tascap_doit, 691
Tasseled Cap Transform, 691
Test Images, 557
Texture Co-occurrence, 694
Texture Filters
Co-occurrence, 694
Texture Measure, 698
Texture Measure, 698
texture_cooccur_doit, 694
texture_stats_doit, 698
Threshold ROIs, 638
Tile Data, 446
Tile Data Initialization, 452
Tile Processing Completed, 532
Tiled Processing, 102, 115
Tiled Processing Increments, 488
tims_cal_doit, 701
TM Calibration, 704, 704
tmcal_doit, 704
Toggle Catch, 31
topo_doit, 708
topo_feature_doit, 712
Topographic Features, 712
Topographic Modeling, 708
Topographic Modeling Tools
topo_doit program, 708
Translate Projection Name, 534
translate projection units, 535

U
unmix_doit, 717
Unmixing, 717
User Defined Move Routine, 148
User Defined Projection Types, 142
User Defined Units, 147
User Functions, 29, 201
Adapting for ENVI Runtime, 120
Adding Widgets to, 82
ENVI Compound Widgets, 740
ENVI Subroutines, 423
Introduction, 73
Making a Simple Function, 77
Recovering from a Crash, 79
Trapping Errors, 98
and Widget Event Handlers, 73

V
variables
named, 203, 204
VAX IEEE Floating Point Conversion, 720
vax_ieee_doit, 720

W
Warping Images, 619
Widget
Adding to User Functions, 82
Arrow Toggle, 767
Auto-Based, 722
Auto-Managed Events, 92
Base, 724
Controlling, 729
Creating a Menu, 742
Creating Lists, 758
Displaying Scroll Bar Labels, 757
Editing Multiple List Values, 735
Endmember Collection, 182
Entering a Parameter, 750
Entering a String, 763
Event Handlers, 73
Glossary of, 83
Illustrations, 83
Input/Edit Map Coordinates, 740
Modifying RGB Values, 755
Pulldown Menu, 753
Selecting Multiple Items, 744
Selecting Output File, 748
Selecting Output Filename, 746
Setting Slider Values, 760
Specifying Image Subsets, 765
Specifying Latitude and Longitude Values, 738
widget_auto_base, 722
widget_base, 724
widget_control, 729
widget_edit, 735
widget_geo, 738
widget_map, 740, 740
widget_menu, 742
widget_multi, 744
widget_outf, 746
widget_outfm, 748
widget_param, 750
widget_pmenu, 753
widget_rgb, 755
widget_slabel, 757
widget_slist, 758
widget_sslider, 760
widget_string, 763
widget_subset, 765
widget_toggle, 767
Writing Batch Routines, 65

X
XSTART, 168

Y
YSTART, 168