

# ER Mapper

Customization Guide

September 2008

*This space reserved for graphic.*

*Left and right bleed.*



Copyright © 2008 ERDAS, Inc.

All rights reserved.

Printed in the United States of America.

The information contained in this document is the exclusive property of ERDAS, Inc. This work is protected under United States copyright law and other international copyright treaties and conventions. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, except as expressly permitted in writing by ERDAS, Inc. All requests should be sent to the attention of:

Manager, Technical Documentation  
ERDAS, Inc.  
5051 Peachtree Corners Circle  
Suite 100  
Norcross, GA 30092-2500 USA.

The information contained in this document is subject to change without notice.

**Government Reserved Rights.** MrSID technology incorporated in the Software was developed in part through a project at the Los Alamos National Laboratory, funded by the U.S. Government, managed under contract by the University of California (University), and is under exclusive commercial license to LizardTech, Inc. It is used under license from LizardTech. MrSID is protected by U.S. Patent No. 5,710,835. Foreign patents pending. The U.S. Government and the University have reserved rights in MrSID technology, including without limitation: (a) The U.S. Government has a non-exclusive, nontransferable, irrevocable, paid-up license to practice or have practiced throughout the world, for or on behalf of the United States, inventions covered by U.S. Patent No. 5,710,835 and has other rights under 35 U.S.C. § 200-212 and applicable implementing regulations; (b) If LizardTech's rights in the MrSID Technology terminate during the term of this Agreement, you may continue to use the Software. Any provisions of this license which could reasonably be deemed to do so would then protect the University and/or the U.S. Government; and (c) The University has no obligation to furnish any know-how, technical assistance, or technical data to users of MrSID software and makes no warranty or representation as to the validity of U.S. Patent 5,710,835 nor that the MrSID Software will not infringe any patent or other proprietary right. For further information about these provisions, contact LizardTech, 1008 Western Ave., Suite 200, Seattle, WA 98104.

ERDAS, ERDAS IMAGINE, IMAGINE OrthoBASE, Stereo Analyst and IMAGINE VirtualGIS are registered trademarks; IMAGINE OrthoBASE Pro is a trademark of ERDAS, Inc.

SOCET SET is a registered trademark of BAE Systems Mission Solutions.

Other companies and products mentioned herein are trademarks or registered trademarks of their respective owners.

# Table of Contents

Table of Contents . . . . .	iii
Preface . . . . .	1
Using the ER Mapper library . . . . .	3
<b>Windows applications</b> . . . . .	<b>3</b>
Linking to MSVCRT . . . . .	4
Adding support for new file formats . . . . .	5
File Syntax . . . . .	7
<b>Data and processing files</b> . . . . .	<b>7</b>
Data files . . . . .	7
Dynamic link specifications . . . . .	8
Processing and configuration files . . . . .	8
Common elements of ER Mapper files . . . . .	10
<b>An example file</b> . . . . .	<b>13</b>
Integrating data with ER Mapper . . . . .	15
<b>Introduction</b> . . . . .	<b>15</b>
ER Mapper Raster File format . . . . .	15
ER Mapper Vector File format . . . . .	16
Dynamic Links . . . . .	16
Vector images and Dynamic Links . . . . .	18
Open Standards . . . . .	18
Testing . . . . .	18
Looking ahead . . . . .	18
<b>Supplied Dynamic</b> . . . . .	<b>19</b>
Example Dynamic Links . . . . .	20
Raster Datasets and Header Files (.ers) . . . . .	21
<b>Introduction</b> . . . . .	<b>21</b>
The header file . . . . .	21
The data file . . . . .	21
<b>The raster header file</b> . . . . .	<b>21</b>
<b>DatasetHeader Block</b> . . . . .	<b>21</b>
Compulsory entries . . . . .	22
Optional entries . . . . .	23
<b>Coordinate Space Block</b> . . . . .	<b>24</b>
Compulsory entries . . . . .	24
Optional entries . . . . .	25
<b>RasterInfo Block</b> . . . . .	<b>26</b>
Compulsory entries . . . . .	26
Optional entries . . . . .	27
<b>CellInfo Block</b> . . . . .	<b>28</b>

<b>Registration Coord Block</b> . . . . .	<b>28</b>
<b>BandId Block</b> . . . . .	<b>29</b>
Compulsory entries . . . . .	29
Optional entries . . . . .	29
<b>SensorInfo Block</b> . . . . .	<b>29</b>
Compulsory entries . . . . .	30
Optional entries . . . . .	30
<b>FiducialInfo Block</b> . . . . .	<b>31</b>
Fiducial point block . . . . .	31
<b>RegionInfo Block</b> . . . . .	<b>31</b>
<b>WarpControl Block</b> . . . . .	<b>32</b>
<b>Correction Block</b> . . . . .	<b>35</b>
<b>GivenOrthoInfo Block</b> . . . . .	<b>36</b>
<b>FFTInfo Block</b> . . . . .	<b>37</b>
<b>Example dataset header files</b> . . . . .	<b>38</b>
Example Landsat header file . . . . .	38
Georeferenced dataset header file . . . . .	39
Example of a complex dataset header file . . . . .	39
Raster data file . . . . .	44
 Vector Datasets and Header Files (.erv) . . . . .	 <b>47</b>
<b>Introduction</b> . . . . .	<b>47</b>
The header file . . . . .	47
The data file . . . . .	47
<b>Vector header file format</b> . . . . .	<b>48</b>
<b>DatasetHeader Block</b> . . . . .	<b>49</b>
Compulsory entries . . . . .	49
Optional entries . . . . .	50
<b>CoordinateSpace Block</b> . . . . .	<b>51</b>
Compulsory entries . . . . .	51
Optional entries . . . . .	52
<b>VectorInfo Block</b> . . . . .	<b>52</b>
<b>Extents Block</b> . . . . .	<b>53</b>
<b>Example vector header files</b> . . . . .	<b>53</b>
Example "RAW" vector dataset header file . . . . .	53
Vector data file . . . . .	54
 Dynamic Links Program Interface . . . . .	 <b>59</b>
<b>Introduction</b> . . . . .	<b>59</b>
Non-Raster data . . . . .	59
ER Mapper PostScript . . . . .	59
Editable links . . . . .	59
Flexible connections . . . . .	59
Multiple links . . . . .	59
Ready-made links . . . . .	59
Triggering links . . . . .	59
Examples . . . . .	59

<b>Georeferenced and Page Relative layers</b> . . . . .	<b>60</b>
Georeferenced data . . . . .	60
Page relative layers . . . . .	60
Dynamic Link layer buttons . . . . .	60
Inside a Dynamic Link . . . . .	62
<b>Creating a Dynamic Link</b> . . . . .	<b>64</b>
ER Mapper PostScript . . . . .	64
Looking ahead . . . . .	64
Adding the menu option . . . . .	65
<b>Chooser program</b> . . . . .	<b>66</b>
Name and Directory . . . . .	66
Arguments . . . . .	66
Output . . . . .	66
Link initialization . . . . .	67
<b>Example initialization program</b> . . . . .	<b>69</b>
<b>PostScript generation</b> . . . . .	<b>70</b>
Name and Directory . . . . .	70
Arguments . . . . .	70
Output . . . . .	71
Errors . . . . .	71
<b>Printing dynamic links.</b> . . . . .	<b>72</b>
<b>MONOCOLOUR vs TRUECOLOUR links</b> . . . . .	<b>73</b>
<b>Batch Script Chooser.</b> . . . . .	<b>73</b>
<b>Debugging dynamic lines.</b> . . . . .	<b>73</b>
Example 1 - Example User Dynamic Link . . . . .	75
Example 2 - Table of data shown as circles . . . . .	78
Example 3 - Dynamic Link to ARC/INFO . . . . .	89
<b>PostScript</b> . . . . .	<b>95</b>
Dimensions in pixels . . . . .	95
Variable canvas size . . . . .	95
<b>Defined variables</b> . . . . .	<b>95</b>
Example . . . . .	96
ER Mapper PostScript . . . . .	96
Standard Postscript . . . . .	96
Variable aspect ratio . . . . .	97
Dimensions in points . . . . .	97
<b>Summary of ER Mapper PostScript functions</b> . . . . .	<b>98</b>
Example . . . . .	99
<b>Performance and accuracy</b> . . . . .	<b>100</b>
Use device DPI . . . . .	100
Integers not reals . . . . .	100
Reduce PostScript file size . . . . .	100
Use Monocolour PostScript . . . . .	101
Extract current data . . . . .	101
Use ER Mapper coordinates . . . . .	101
<b>Using existing PostScript</b> . . . . .	<b>101</b>
<b>ervecmacro.erm</b> . . . . .	<b>101</b>

The ER Mapper Configuration File (config.erm) . . . . .	103
<b>ERMConfig Block</b> . . . . .	<b>103</b>
Algorithm files (.alg) . . . . .	105
<b>Example algorithm file</b> . . . . .	<b>105</b>
<b>Algorithm file entries</b> . . . . .	<b>113</b>
<b>Coordinate space block</b> . . . . .	<b>115</b>
<b>TopLeftCorner and BottomRightCorner Blocks</b> . . . . .	<b>116</b>
<b>PageSize Block</b> . . . . .	<b>116</b>
<b>ThreeDInfo block</b> . . . . .	<b>118</b>
<b>Surface Blocks</b> . . . . .	<b>120</b>
<b>Stream Blocks</b> . . . . .	<b>121</b>
<b>StreamInput Sub-Blocks</b> . . . . .	<b>123</b>
<b>Kernel Sub-Blocks</b> . . . . .	<b>123</b>
<b>Formula Sub-Blocks</b> . . . . .	<b>123</b>
<b>Transform Sub-Blocks</b> . . . . .	<b>124</b>
<b>Dynamic Link Overlay Entries</b> . . . . .	<b>125</b>
Filter files (.ker) . . . . .	127
<b>C Filters</b> . . . . .	<b>129</b>
Formula files (.frm) . . . . .	131
<b>FormulaArg Blocks</b> . . . . .	<b>131</b>
<b>BandID Blocks</b> . . . . .	<b>132</b>
C filters and functions . . . . .	135
<b>Filters/kernels</b> . . . . .	<b>135</b>
<b>Writing and compiling usercode on PCs</b> . . . . .	<b>135</b>
Limitations . . . . .	135
Setting and referring to Windows environment variables . .	136
Checklist . . . . .	136
Example code . . . . .	137
Building new formula and filters manually . . . . .	137
To edit formula and kernels/filters C source code . . . . .	137
To write or build your own formula manually . . . . .	137
To write and build your own filter/kernel manually . . . . .	138
To compile usercode directories (formulae, filters/kernels)	138
To manually compile usercode . . . . .	139
Notes on compiling libermapper programs on PC . . . . .	140
<b>Filter file format</b> . . . . .	<b>142</b>
<b>The filter Object file</b> . . . . .	<b>143</b>
<b>Formula functions</b> . . . . .	<b>143</b>
C user defined function name . . . . .	144
Number of Arguments . . . . .	144

Type of Arguments . . . . .	144
Examples . . . . .	144
<b>C function format . . . . .</b>	<b>144</b>
<b>Initialization and finalization . . . . .</b>	<b>145</b>
Filters . . . . .	145
Formula . . . . .	146
<b>Linking . . . . .</b>	<b>147</b>
Default switches . . . . .	147
Special case linking . . . . .	147
<b>Special C Function arguments . . . . .</b>	<b>147</b>
INPUTVECTOR . . . . .	148
USERSTATS . . . . .	148
<b>Creating Color Tables . . . . .</b>	<b>151</b>
To select a Lookup Table . . . . .	151
To create a new Lookup Table . . . . .	151
To generate a linear Color Lookup Table . . . . .	151
Example: a 2-color Lookup Table . . . . .	152
<b>Example: a multicolor Lookup Table . . . . .</b>	<b>152</b>
<b>Example: assigning specific colors to each grey level . . . . .</b>	<b>152</b>
<b>Look Up Table files (.lut) . . . . .</b>	<b>155</b>
<b>LUT file entries . . . . .</b>	<b>155</b>
<b>Hardcopy files (.hc) . . . . .</b>	<b>157</b>
<b>Menu and toolbar files (.erm) and (.bar) . . . . .</b>	<b>161</b>
<b>Submenus . . . . .</b>	<b>162</b>
<b>Menu option keyboard selection . . . . .</b>	<b>163</b>
<b>Option separators . . . . .</b>	<b>163</b>
<b>Available callback options . . . . .</b>	<b>163</b>
<b>Map Composition files (.ltd) . . . . .</b>	<b>167</b>
<b>LegendRule block . . . . .</b>	<b>169</b>
<b>UserParameter Block . . . . .</b>	<b>170</b>
<b>Allowable Parameter types . . . . .</b>	<b>171</b>
Algorithm . . . . .	171
YES/NO . . . . .	171
String . . . . .	171
Number . . . . .	171
List . . . . .	172
Color . . . . .	172
Font . . . . .	172
DynamicLink . . . . .	174
<b>Generate Block . . . . .</b>	<b>174</b>
AlgorithmImagePS . . . . .	174

Progirst . . . . .	174
Linepath . . . . .	174
Boundingbox . . . . .	174
ScaleBar . . . . .	175
Special User Parameter "GeodeticScaleLatitude" for scale bars . . . . .	176
ZScale . . . . .	176
NorthArrow . . . . .	176
Grid . . . . .	176
<b>PSLibInclude Block . . . . .</b>	<b>177</b>
<b>PSExecInclude Block . . . . .</b>	<b>177</b>
<b>Example . . . . .</b>	<b>178</b>
Dynamic Links . . . . .	183
<b>Menu entries . . . . .</b>	<b>183</b>
Submenus . . . . .	185
<b>Menu entry parameters . . . . .</b>	<b>185</b>
To add a Dynamic Link option . . . . .	186
<b>Link chooser parameter . . . . .</b>	<b>187</b>
Digitizer files . . . . .	191
<b>Digitizer type file .dtp . . . . .</b>	<b>191</b>
Notes . . . . .	192
Format of .dtp files. . . . .	192
Calcomp digitizers . . . . .	193
GTCO digitizers . . . . .	193
Altek digitizers . . . . .	193
<b>Digitizer configuration file .dcf . . . . .</b>	<b>194</b>
<b>Digitizer session file .dig . . . . .</b>	<b>195</b>
<b>Digitizer modes . . . . .</b>	<b>196</b>
Hardcopy Processing and Filter programs . . . . .	197
<b>Hardcopy processing stages . . . . .</b>	<b>197</b>
Create device independent output . . . . .	197
Convert to device format . . . . .	197
Send to device . . . . .	197
Strip printing . . . . .	198
<b>The hardc . . . . . opy engine</b>	<b>198</b>
The hardcopy engine Output Format . . . . .	199
<b>Filter Program . . . . .</b>	<b>199</b>
Creating a Filter Program (.c) . . . . .	199
<b>Color compression . . . . .</b>	<b>209</b>
<b>Dithering . . . . .</b>	<b>209</b>
<b>Image compression . . . . .</b>	<b>210</b>
<b>Chroma correction . . . . .</b>	<b>210</b>
<b>Output Program . . . . .</b>	<b>210</b>



<b>Subsampling/Supersampling</b> . . . . .	<b>211</b>
<b>Aspect ratio</b> . . . . .	<b>211</b>
<b>Hardcopy output size</b> . . . . .	<b>211</b>
<b>Color</b> . . . . .	<b>211</b>
Importing using a command line . . . . .	213
<b>Raster and vector image formats</b> . . . . .	<b>213</b>
<b>Raster File Import Switches</b> . . . . .	<b>214</b>
<b>Importing Raster File examples</b> . . . . .	<b>216</b>
Example 1 . . . . .	216
Example 2 . . . . .	216
Example 3 . . . . .	217
Example 4 . . . . .	217
Example 5 . . . . .	217
Example 6 . . . . .	217
Example 7 (importusgs) . . . . .	218
<b>Vector File Import Switches</b> . . . . .	<b>218</b>
<b>Importing Vector File examples</b> . . . . .	<b>219</b>
Example 1 . . . . .	219
Example 2 . . . . .	219
<b>Invert</b> . . . . .	<b>219</b>
Utilities . . . . .	221
<b>Coordinate conversion</b> . . . . .	<b>221</b>
togeo . . . . .	221
fromgeo . . . . .	221
gdt_conv . . . . .	221
<b>Tape utilities</b> . . . . .	<b>221</b>
erm_tapeutil . . . . .	221
tape_struct . . . . .	222
dump_cct . . . . .	222
tape2disk . . . . .	222
disk2tape . . . . .	222
<b>Image file compression</b> . . . . .	<b>222</b>
ecw_compress . . . . .	222
ecw_compress_gui . . . . .	223
Batch scripting and wizards . . . . .	225
<b>Batch script documentation</b> . . . . .	<b>225</b>
<b>Creating a batch script</b> . . . . .	<b>226</b>
Getting started . . . . .	226
Example display and geolinking script . . . . .	229
<b>Wizard scripts</b> . . . . .	<b>233</b>
Example wizard script . . . . .	234
The layout of wizard pages . . . . .	236
General guidelines for wizard pages . . . . .	239
Example wizard . . . . .	240

Scripting language . . . . .	247
<b>Input to batch process.</b> . . . . .	<b>247</b>
Command line arguments . . . . .	248
Dialog boxes . . . . .	248
Dialog box input fields . . . . .	249
Wizards . . . . .	250
Using preferences to remember settings . . . . .	251
<b>ER Mapper Objects.</b> . . . . .	<b>253</b>
Image Manipulation . . . . .	254
Actions . . . . .	257
Attributes . . . . .	258
Command summaries . . . . .	261
Preferences . . . . .	277
File and Path separators . . . . .	277
Other commands . . . . .	278
<b>Output.</b> . . . . .	<b>279</b>
Output to image window . . . . .	280
Output to file . . . . .	280
Output to Batch Engine Output dialog (print commands) . . . . .	280
Warning dialog . . . . .	281
Status dialog . . . . .	281
<b>Library of batch scripts . . . . .</b>	<b>282</b>
Scripting reference . . . . .	283
<b>Operators . . . . .</b>	<b>283</b>
Concatenation . . . . .	283
<b>Mathematical functions . . . . .</b>	<b>283</b>
<b>Variables . . . . .</b>	<b>284</b>
Arrays . . . . .	287
<b>Page size options . . . . .</b>	<b>288</b>
<b>Keywords . . . . .</b>	<b>288</b>
<b>Flow control . . . . .</b>	<b>290</b>
Labels . . . . .	290
Controls . . . . .	290
Explicit exit . . . . .	290
Looping . . . . .	291
<b>Including files. . . . .</b>	<b>291</b>
<b>Error reporting . . . . .</b>	<b>291</b>
<b>Script Commands - Alphabetical listing . . . . .</b>	<b>291</b>
Example . . . . .	302
Examples . . . . .	305
Example . . . . .	329
Example . . . . .	337
Sensor Platform Characteristics . . . . .	359
<b>NOAA . . . . .</b>	<b>359</b>
<b>Geoscan MSS Mark 2 . . . . .</b>	<b>360</b>

<b>IRS-IC</b> . . . . .	<b>364</b>
<b>LANDSAT 4 or LANDSAT MSS</b> . . . . .	<b>367</b>
<b>LANDSAT 5 or LANDSAT TM</b> . . . . .	<b>369</b>
<b>LANDSAT 7</b> . . . . .	<b>372</b>
<b>SPOT</b> . . . . .	<b>372</b>
Data suppliers . . . . .	377
Australian Centre for Remote Sensing (ACRES) . . . . .	377
Australian Geological Survey Organisation (AGSO) . . . . .	377
Department of Mineral Resources TAS . . . . .	377
Earth Observation Satellite Company . . . . .	378
Eurimage . . . . .	378
GeoImage Pty Ltd . . . . .	378
Intera Information Technologies Corporation . . . . .	379
Kevron Geophysics Pty Ltd . . . . .	379
National Geophysical Data Center . . . . .	379
RADARSAT International, Inc. . . . .	379
Satellite Remote Sensing Services . . . . .	380
SPOT Imaging Services Pty Ltd . . . . .	380
SSC Satellitbild . . . . .	381
USGS . . . . .	381
Index . . . . .	383



# Preface

This section contains information for C/C++ programmers interested in working with the ER Mapper library or in adding to the file formats that ER Mapper can directly open or save. In particular, it contains information on linking applications with the ER Mapper library that will be of use to those developers who wish to update applications linking with the library.

For more detailed information on any of these topics, please contact your regional ER Mapper office.



# Using the ER Mapper library

The ER Mapper library contains powerful functions for reading and writing ER Mapper image datasets and algorithms. This chapter provides information on how to link the ER Mapper library with your application.

For more detailed information on using the ER Mapper library contact your regional ER Mapper office and ask them about the ER Mapper Software Developers' Kit (the ER Mapper SDK).

## Windows applications

The following is a description of how to set up your project to enable you to use the ER Mapper library with your application. Note that you must have the following installed:

- ER Mapper
- Microsoft Visual Studio 2005



*In the following **<ermapperdir>** is the directory in which ER Mapper was installed. This is often **C:\Program Files\ERDAS\ERDAS ER Mapper 7.2**.*

1. Set up the following environment variables:

ERMAPPER	Should point to <b>&lt;ermapperdir&gt;</b> .
ERM_MACHINE_TYPE	Should be " <b>win32</b> ".
PATH	You need to add <b>&lt;ermapperdir&gt;\bin\win32</b> to your PATH variable.
ERMTMP	Should point to a directory which can be used to store temporary files (optional)

2. Start Microsoft Visual Studio 2005 and load your project workspace.
3. Select **Tools->Options** from the menu and do the following:
  - a) Select the **Directories** tab and then select **Library files** from the **Show directories for** list. Add:  
`<ermapperdir>\lib\win32`
  - b) Select **Include files** from the **Show directories for** list. Add the following:  
`<ermapperdir>\include`
4. Select **OK**.
5. Select **Project->Settings** from the menu.
6. Select **Win32 Debug** from the **Settings for** list.

a) Select the **C/C++** tab and add the following to the **Preprocessor definitions** field:

```
,win32,INC_OLE1.
```

b) Select **Code Generation** in the Category list and change the **Use run-time library** field to **Multithreaded DLL**.

c) Select the **Link** tab and add the following to **Object/library modules** field:

```
ermapper.lib wsock32.lib.
```

d) Select **Input** from the **Category** list. To the **Ignore Libraries** field add:

```
libc.lib
```

7. Select **OK**.
8. Repeat step 6, but select **Win32 Release** instead of **Win32 Debug** from the **Settings For** list.
9. Select the **Project->Build projectname** option from the menu. The project should now compile and link.
10. To run the examples from within the Visual Studio 2005 environment, select **Project->Settings...** from the menu, then select the **Debug** tab, and then enter your program arguments (if any). You can then select **Build->Execute projectname** from the menu to run the program.

## Linking to MSVCRT

When you are linking or creating raster translators, you should try to link with the same version of MSVCRT, and in the same way that ER Mapper is linked. If your code installs its own instance of MSVCRT it also creates a second malloc heap. The ER Mapper DLL could then free up memory allocated by your copy of MSVCRT.



# Adding support for new file formats

The list of file formats which can be opened or saved by ER Mapper is extensible through the addition of raster translators which can be written by anyone. Raster translators are dynamic linked libraries (DLLs) on Windows systems..

Once written, raster translators must be installed in the raster\_translator directory of the product using the ER Mapper library. With ER Mapper and ER Viewer, this is the **raster\_translators\win32** directory.



# File Syntax

ER Mapper has a number of internal files and file formats. You don't generally need to know about them because they are transparent when you use ER Mapper. They are included here for advanced users and for trouble-shooting. ER Mapper has a policy of open standards so all formats are included here.

This chapter introduces the file formats including the file extensions and common elements. Read this chapter first so you are familiar with the conventions in the rest of the manual, then turn to the chapter dedicated to the file format you are interested in.

- Part 1 discusses the different ways of integrating data with ER Mapper by importing or using dynamic links. The raster and vector formats and dynamic link specifications are detailed in this part.
- Part 2 documents the remaining types of processing and configuration files that ER Mapper uses.
- Part 3 gives the syntax for carrying out selected ER Mapper processing from the command line.

## Data and processing files

ER Mapper files can be divided into two general types: data files, and processing and configuration files. Data files store the data in internal raster and vector file formats. Processing and configuration files define the processing you want to apply and store processing control parameters.

### Data files

ER Mapper has internal raster and vector file formats. In general we refer to data as though they were kept in individual files. Most of the time this is how we use them. In reality, however, all data files are made up of two parts: the data itself and an associated header file. The data file holds just the data—pixel values for raster data in binary format and object vertices and drawing information for vector data in ASCII format. The header file contains information about the data, for example, how many rows and columns there are, what the map projection is, etc. Raster and vector data file names do not have file extensions, their corresponding header files have the same name as the data file and with **.ers** and **.erv** extensions respectively. Data files can have a different name to their corresponding header files as long as data file name is defined in the header file.

File Type	File Extension	File Function
Raster data	none	Binary raster image data file (requires an accompanying <b>.ers</b> file). For example, <b>Australia_DTM</b>

File Type	File Extension	File Function
Raster data header	<b>..ers</b>	ASCII (text) file. Identifies characteristics of a raster image file. For example, <b>Australia_DTM.ers</b>
Vector data	none	ASCII (text) vector data file (requires an accompanying <b>.erv</b> file). For example, <b>San_Diego_Roads</b>
Vector data header	<b>..erv</b>	ASCII (text) file. Identifies characteristics of a vector data file. For example, <b>San_Diego_Roads.erv</b>



*ER Mapper can also add a **.ers** header file to ECW compressed image data files and to those of other supported formats like GeoTIFF/TIFF and JPEG.*

### Dynamic link specifications

In addition to internal data file formats, ER Mapper can link to external data formats through its dynamic link capability. This gateway displays any (properly formatted) PostScript information. Included with ER Mapper are dynamic links to Geographic Information Systems (ARC/INFO), databases (Oracle) and external file formats (.dxf files etc). Links to any PostScript data can be added using dynamic links.

### Processing and configuration files

ER Mapper stores processing commands in ASCII (plain text) files which can be easily read and edited. Most of the ASCII files are created automatically by ER Mapper as you use the Graphical User Interface to request different image processing tasks. ER Mapper also uses information in configuration files which are set up to customize it.

Each of the different file types is distinguished by its file name extension. They are listed below with a brief description of the function of each file type.

File Type	File Extension	File Function
Configuration	config.erm	Holds license details, computer display attributes etc
Algorithm	.alg	Describes processing tasks to be applied to one or more input data sources to produce an image display
Filter/Kernel	.ker	Describes the values for a 2-D filter/convolution kernel

<b>File Type</b>	<b>File Extension</b>	<b>File Function</b>
Formula	.frm	Describes a mathematical formula to be applied to one or more input sources to produce one layer of an image display
Lookup Table	.lut	Defines the values for a color lookup table
Hardcopy	.hc	Defines the capabilities of a hardcopy device
Menu	.erm	Define the ER Mapper menus
ervecmacro	.erm	Internal Read Only file converts ER Mapper vector files to PostScript
Batch	.erb	Files written in ER Mapper batch scripting language.
Camera	.cam	Holds camera details required for orthorectification.
Grid Project	.egp	Project File created and used by the Gridding Wizard.

Each of the file types is discussed in detail in chapters which follow in this manual. The remainder of this chapter describes terminology and structures common to all the ER Mapper files.

## Common elements of ER Mapper files

ER Mapper ASCII files are made up of lines of text which specify parameters. These parameters define characteristics and processing; some are compulsory, others are optional. Related parameter entries are grouped together in *blocks*. The examples in this chapter refer to a dataset header file but the same block structure and parameter specification syntax is used in all (non-data) files. A simple example of a dataset header file is given below. The entries in this file are explained in ["Raster Datasets and Header Files \(.ers\)"](#) in this manual. For the moment just note the block structure and parameter specification syntax; these are explained in the remainder of the chapter.

```
DatasetHeader Begin
  Version = "7.2"
  DataSetTypen= ERStorage
  DataTypen = Raster
  ByteOrder = MSBFirst
  CoordinateSpace Begin
    Datum = "RAW"
    Projection = "RAW"
    CoordinateType= RAW
    Rotation = 0:0:0.0
  CoordinateSpace End
  RasterInfo Begin
    CellType = IEEE4ByteReal
    NrOfLines = 401 # Number of lines
    NrOfCellsPerLine= 300 # Number of cells per line
    NrOfBands = 3# Number of bands
  RasterInfo End
DatasetHeader End
```

### Comment lines

Blank lines are disregarded. The hash (#) symbol identifies a line or part of a line as a comment. Characters following the '#' through to the end of the line are ignored. For example, the following line:

```
Xdimension = 10.0 # the cell size is in metres
```

will be read by ER Mapper as:

```
Xdimension = 10.0
```

### Parameter Specification entries

ER Mapper files are made up of parameter specification entries which assign a value to one of the parameters used by ER Mapper. The syntax is:

*parameter\_name* = *value*

For example,

```
NrOfCellsPerLine = 1000
Projection = "RAW"
```

**Parameter Names.** Parameter names are usually made up of several words run together, with the first character of each word capitalized. This capitalization is to make the parameters easier to read and is optional. ER Mapper ignores the character case when it processes the entry lines. For example, `NrOfCellsPerLine`, `NROFCELLSPERLINE` and `nrofcellspersline` are all equivalent. Required and optional parameter and syntax are documented for each of the ER Mapper file types in the relevant chapters in this manual.

**Values.** There are a number of different types of values used by ER Mapper. These are listed below.

- **text strings:** These are enclosed in double quotes and are case sensitive (upper and lower case are not equivalent). Often strings are used to display information in a message window or description field while running ER Mapper. For example:

```
Name = "wetlands_scene"
```

Embedded quotes should be preceded by a backslash (\). For example,

```
Description = "A \"sample\" name"
```

- **integers:** These are whole numbers such as -3, 0, 2. For example:

```
NumberOfBands = 5
```

- **real numbers:** Floating point or real numbers may or may not have a decimal point, for example 43 or 1.2353234 are both valid. Exponent form, for example 1.313E+02 is also allowed. For example:

```
NullCellValue = -999
```

```
NullCellValue = 123.456
```

```
NullCellValue = 1.0E+26
```

- **arrays:** These can be arrays of integers or real numbers. They are entered within curly braces:

```
parameter = {  
  . . . list of entries. . .  
}
```

For example:

```
SubRegion = {  
  0 0  
  0 1851  
  15361851  
  15360  
}
```

- **dates:** Times are specified in GMT (Greenwich Mean Time) with the following syntax:

*Day Month DayOfMonth Hrs:Mins:Secs GMT Yr*

For example:

`SenseDate = Sun Dec 7 03:20:25 GMT 1989`

- **angles:** Angles are specified in *Degrees:Minutes:Seconds* with Degrees and Minutes in integers and Seconds in decimal seconds. For example,

`Rotation = 23:45:34.6`

- **keywords:** A keyword entry requires you to select one of a number of defined options. These options are described in the relevant chapters in the remainder of this manual. Keywords are used without quotation marks and are case insensitive (upper and lower case is disregarded). For example, the available options for the CellType above are:

- Unsigned8BitInteger
- Signed8BitInteger
- Unsigned16BitInteger
- Signed16BitInteger
- Unsigned32BitInteger
- Signed32BitInteger
- IEEE4ByteReal
- IEEE8ByteReal

and an example entry is:

`CellType = Unsigned8BitInteger`



## Begin-End Blocks

The parameter specification entries are grouped into blocks of related definitions. Each block is enclosed by a "*block\_name* Begin" statement and a "*block\_name* End" statement. For example, the major block which describes the contents of a dataset header file is called the DatasetHeader block and all the entries are typed between the two lines below.

```
DatasetHeader Begin
.
. body of block
.
DatasetHeader End
```

The entries within a block are indented to make it easier to see the block structure. ER Mapper indents the entries when it creates or modifies a file; however the indentation is optional. If you modify a file and do not use indentation ER Mapper will still read and process the file.

Blocks can contain one or more sub-blocks with more specific information about an aspect of the larger block. For example, the DatasetHeader block contains the sub-block RasterInfo which describes the cell (pixel) size and image size (in cells). RasterInfo, in turn, contains multiple entries of the sub-block BandId which describe the bands or sensor channels contained in the file.

The order of the parameter specification entries within blocks and sub-blocks and the order of sub-blocks within a block is arbitrary.

## An example file

An example of a simplified raster header file is shown below.

```

# My simplified header file (this comment line will be ignored by ER Mapper)
# Note: This is for illustration only.
# All characters on a line after the '#' are ignored (comments)
#
DataSetHeader Begin                # start of block
  LastUpdated = Tue Sep 3 03:20:25 GMT 1998# a time entry
  SensorName = "2-band multispectral"    # a string
  SenseDate = Sun Nov 30 09:15:25 GMT 1989# a time entry
  DataSetType = ERStorage # A keyword entry (see above)
  DataType = Raster    # A keyword entry
  ByteOrder = MSBFirst # A keyword entry
  CoordinateSpace Begin # Sub block of the DataSetHeader
    Datum = "RAW"      # String
    Projection = "RAW" # String
    CoordinateSystem = Raw # A keyword entry
    Rotation = 12:23:34.2# An angle entry
  CoordinateSpace End # Closes or ends this block
  RasterInfo Begin    # Sub block of the DataSetHeader
    CellType = Unsigned8BitInteger # A keyword entry
    CellInfo Begin    # Sub block of the RasterInfo
      XDimension = 10.0 # Real or floating
      YDimension = 10.0 # point entries
    CellInfo End      # Closes or ends this block
    NrOfLines = 956   # Integer number (of lines)
    NrOfCellsPerLine = 758 # Integer number (cells per line)
    NrOfBands = 2     # Integer number (of bands)
    BandId Begin      # Sub block of RasterInfo
      Value = "0.123" # String (band pass in um)
      Width = 0.42    # A real number (band width)
      Units = "um"    # Units applicable to values
    BandId End        # End of first band descriptor
    BandId Begin      # Start of another BandId
                        # (Have two bands in data file)
      Value = "0.234" # String (width of second band)
      Width = 0.21    # Second band width
      Units = "um"    # String describing units
    BandId End        # Close the second block
  RasterInfo End      # End of RasterInfo block
DataSetHeader End     # End of DataSet definition

```

# Integrating data with ER Mapper

## Introduction

One of the important considerations of any image processing system is the range of image data formats it can access. ER Mapper can read, import and display raster and vector files in a range of popular formats. With its Dynamic Links capability, data from potentially any source can be displayed and integrated into ER Mapper imagery.

## ER Mapper Raster File format

Raster images contain data for entire areas in regular grids. In addition to those in ER Mapper Raster file (**.ers**) format, ER Mapper can process and display raster images in the following formats.

- ER Mapper Compressed Image (.ecw)
- Windows Bitmap (.bmp)
- ESRI BIL and GeoSPOT (.hdr)
- GeoTIFF/TIFF (.tif)
- JPEG (.jpg)
- USGS DOQQ (Grayscale)
- RESTEC/NASDA CEOS (.dat)



---

*It is advisable to periodically check the ER Mapper Web site at <http://www.ermapper.com> for new directly supported raster file formats that have been added by ER Mapper or a third party.*

Other raster formats must first be imported. During import a new file in ER Mapper raster format is created. To return imported files to their original format they must be exported.

External Data Source	Data Accessing Mechanism	Editable
External Raster File Format	Import Raster	YES (header)



---

*It is possible to add direct support for additional file formats by writing a new ER Mapper raster translator. Contact Earth Resource Mapping for more information if you want to know more about writing raster translators.*

**ER Mapper Vector File format**

Vector images contain information on an object by object basis. Vector files can only be edited if they are in ER Mapper Vector or ARC/INFO coverage format. Images in other vector formats must first be imported into ER Mapper format. During import, a new file, which can be fully edited and manipulated, is created. To return imported files to their original format they must be exported.

<b>External Data Source</b>	<b>Data Accessing Mechanism</b>	<b>Editable</b>
External Vector Files	Import Vector	YES

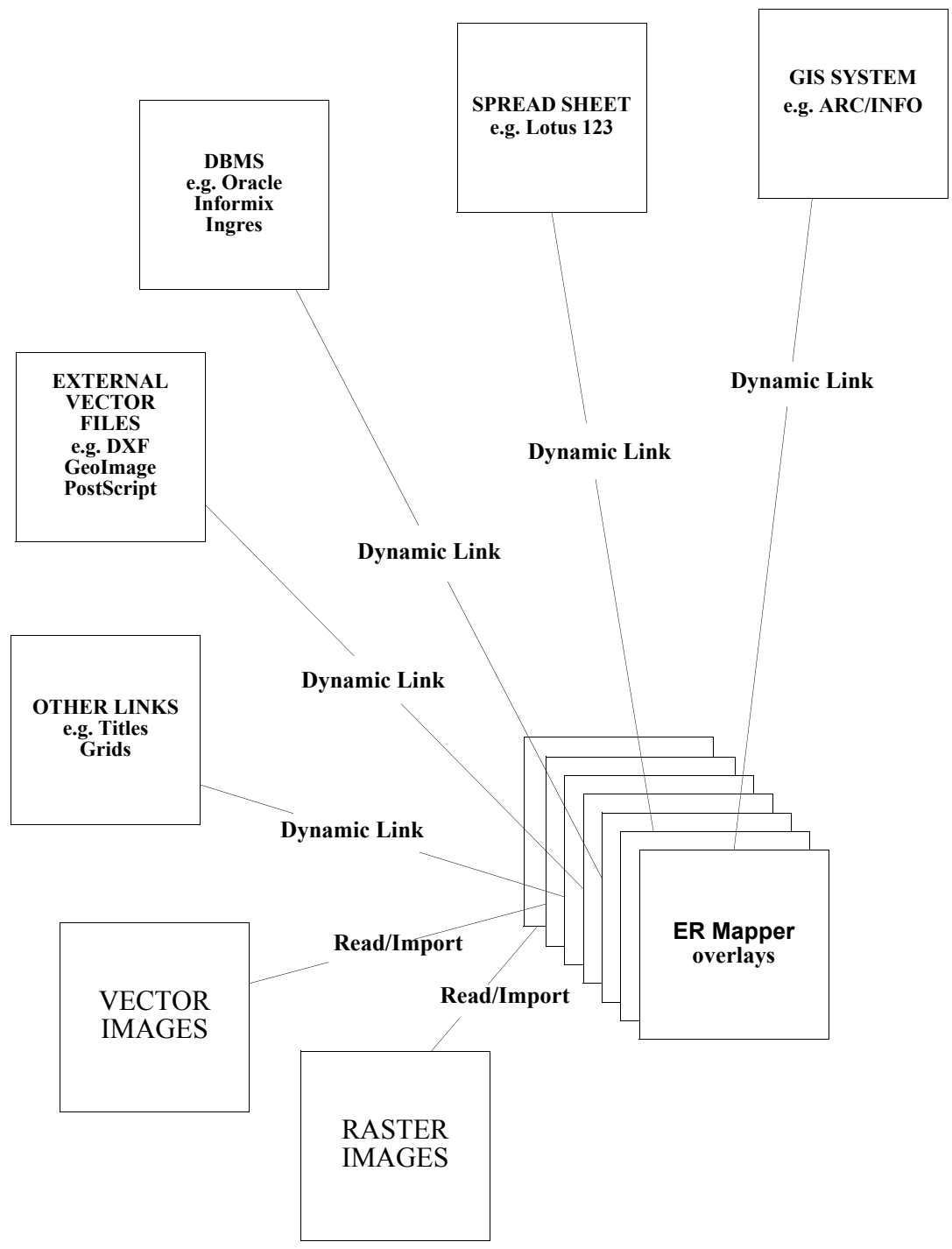
**Dynamic Links**

In addition, there are many other types of data that you might want to incorporate into your display. ER Mapper’s open-ended Dynamic Links provide a gateway through which you can integrate data from virtually any system into your ER Mapper imagery.

The most powerful application of Dynamic Links is to access information in other systems, such as GIS systems, databases and spreadsheets. A simpler Dynamic Link might extract data from a flat file.

Displaying data using Dynamic Links is totally different from importing it. With Dynamic Links the data is not copied into ER Mapper during linking but rather ER Mapper reaches out and uses data which remains in its original system, in its original format. Because ER Mapper works from source data rather than from copies, the image automatically reflects any changes in the source information each time the image is redisplayed.

<b>External Data Source</b>	<b>Data Accessing Mechanism</b>	<b>Editable</b>
GIS systems Database, spreadsheet data System data Generated data External vector files	Dynamic Links	Optional (depends on dynamic link.)



*Displaying data with ER Mapper*

## Vector images and Dynamic Links

Because of the overlap between Dynamic Link and vector capabilities, vector files can be either imported or displayed using a Dynamic Link. The results of the two processes are different. If you want to be able to edit the image using ER Mapper you must import the file into ER Mapper's vector file format. If you don't need to edit the data, display it using a Dynamic Link.

External Data Source	Data Accessing Mechanism	Editable
External Vector Files	Dynamic Links	NO
External Vector Files	Import Vector	YES

## Open Standards

The Dynamic Links, Raster File formats and Vector File formats have been designed to provide a standard interface for building links with other systems. They are open to use by anyone wishing to design a link to ER Mapper: perhaps individuals who want to incorporate data from other systems into their images, or software suppliers who may want to offer their clients a link to ER Mapper or data in ER Mapper format.

## Testing

This section contains all the information required to create a new Dynamic Link or import or export program.

### Programming support

To create a new Dynamic Link you will need to know about the following:

- the C programming language
- PostScript
- the system or data to be linked to
- ER Mapper Dynamic Link mechanism.

To create a new import or export utility you will need to know about:

- the C programming language
- the data format you want to import or export
- ER Mapper Raster or Vector dataset formats.

If you have difficulty in any of these areas please contact Earth Resource Mapping for assistance.

## Looking ahead

The remainder of this section is divided into the following chapters:

**Raster File Format.** This chapter describes the ER Mapper raster dataset file format, consisting of an ASCII ".ers" header file and a binary data file.

**Vector File Format.** This chapter describes the ER Mapper vector dataset file format, consisting of an ASCII “.erv” header file and an ASCII data file.

**Dynamic Links.** This chapter starts by introducing the Dynamic Link facility, using examples to indicate the range of its application. Next, it looks at how Dynamic Links appear to an ER Mapper user and details the programs that are needed to construct a new link. The chapter ends with the complete code for three examples of Dynamic Link.

**PostScript.** This chapter examines how ER Mapper PostScript interpretation differs from standard interpretation.

## Supplied Dynamic

<b>SUPPLIED DYNAMIC LINKS (note that this list is subject to change)</b>	
ARC/INFO Dynamic Link	ARC/INFO coverage
Contours	
External Vector formats	AutoCAD DXF GeoImage MicroStation DGN File Table of data shown as Circles Table circles with Rotation Table of data shown as Outline Circles Grid Datasource Points Grid Datasource TIN
Regions	Regions Outlined Regions Filled

<b>SUPPLIED DYNAMIC LINKS (note that this list is subject to change)</b>	
PostScript	8.5x11 inch Monocolor PostScript 8.5x11 inch Monocolor Compressed PostScript 8.5x11 inch Truecolor PostScript 8.5x11 inch Truecolor Compressed PostScript
Example User Dynamic Links	Example User Dynamic Link Show arguments for No Parameter Show arguments for Dataset Chooser Show arguments for Fixed Parameter Show arguments for \$\$ALG Parameter Show arguments for Link Chooser Show arguments for External Link Chooser Show arguments for Script Link Chooser Dynamic Link coded PostScript

## Example Dynamic Links

### Links to DBMS

Example: Dynamic Link to Oracle (UNIX only)

In this case the data being extracted from the system is tables of figures rather than vector data. The figures are processed and displayed as circles.

### Links to Vector Files

Example: External Vector Formats / AutoCAD DXF

This overlay links to any flat vector file with a ".dxf" extension. This link can be used to access GIS data when a GIS system is not available.

Other links of this type are Dynamic Links to GeoImage and PostScript files. Each link is set up to accept data in the particular external file format.



# Raster Datasets and Header Files (.ers)

## Introduction

An ER Mapper raster dataset is made up of two files:

- the dataset header file, and
- the data file.

## The header file

The dataset header file is an ASCII file describing the raster data in the data file. The dataset header file normally has the same name as the data file that it is describing, with the file extension **".ers"** added. For example, **"Australia\_DTM.ers"** is a valid raster dataset header file name for the raster Data file named **"Australia\_DTM"**. The header file may point to a data file that has a different name via the DataFile field in the header file.



---

*ER Mapper automatically creates a header file for another image format like TIFF to store ancillary information for defining regions etc.*

## The data file

The data file contains the data itself. The raster data is stored in a binary Band-Interleaved-by-Line (BIL) format with a pixel data type defined in the accompanying header file. This format is explained in detail later.

The data file name has no extension. For example, **"Australia\_DTM"** is the raster data file described by the **"Australia\_DTM.ers"** header file.

This chapter details the formats for these two files.

## The raster header file

The entire header file holds information about the data source and is contained in the DatasetHeader Block. There are two compulsory sub-blocks, the CoordinateSpace Block and the RasterInfo Block.

- The CoordinateSpace Block defines the image coordinate space and location.
- The RasterInfo Block defines the characteristics of the data in the accompanying data file and may contain a number of optional sub-blocks.

The next section describes the entries in the DatasetHeader block and the subsequent sections describe sub-blocks.

## DatasetHeader Block

This block contains the whole of the Header file. It has two compulsory sub-blocks: the CoordinateSpace block and the RasterInfo block.

A simple raster header file illustrating the compulsory entries is shown below.

```
DatasetHeader Begin
  Version = "7.2"
  Name = "Australia.ers"
  DataFile = "Shared_Data/Australia"
  DataSetType = ERStorage
  DataType = Raster
  ByteOrder = MSBFirst
  CoordinateSpace Begin
    Datum = "RAW"
    Projection = "RAW"
    CoordinateType = RAW
    Rotation = 0:0:0.0
  CoordinateSpace End
  RasterInfo Begin
    CellType = IEEE4ByteReal
    NrOfLines = 401#Number of lines
    NrOfCellsPerLine = 300#Number of cells/line
    NrOfBands = 3 #Number of bands
  RasterInfo End
DatasetHeader End
```



*Some more complex examples of header files can be found in the ["Example dataset header files"](#) section towards the end of this chapter.*

The entries of a raster dataset header are described below:

### Compulsory entries

- **Version:** Used to define the version. For example,

```
Version = "6.0"
```

- **DataSetType:** The type of image to be displayed. Allowed values are: 'ERStorage' - ER Mapper dataset format  
'Translated' - external (non ER Mapper) format.

```
DataSetType = ERStorage
```

There are many import utilities that may be used to convert images from an external file format into ER Storage format. See ["Vector datasets and header files \(.erv\)"](#).

- **DataType:** `DataType` tells ER Mapper what type of image it is. Values allowed are `Raster` or `Vector`. For example,

```
DataType = Raster
```



Vector image entries are detailed in the ["Vector datasets and header files \(.erv\)"](#) chapter.

- **ByteOrder:** Indicates the byte ordering of the data. Can be either:
  - `MSBFirst` - Most significant byte first.
  - `LSBFirst` - Least significant byte first.

For example,

```
ByteOrder = MSBFirst
```

- **CoordinateSpace block:** Defines the image projection. See ["Coordinate space block"](#) below.
- **RasterInfo block:** Defines the image format. See ["Raster info block"](#) below.

## Optional entries

- **Name:** The name of this header file. ER Mapper inserts or updates this entry whenever it edits the header file. This includes changes made via the Dataset Header Editor, adding regions, calculating statistics, analyzing the image before balancing etc.
- **SourceDataset:** When processing classified datasets, the training region statistics must be calculated from the dataset on which the training regions were originally defined. This is done by using the SourceDataset entry which contains the path/name of the dataset on which the training regions were originally defined, e.g., `c:\imagefiles\someimage.tif`.
- **DataFile:** The name of the data file described by this header file. The defaults to the same name (without the `.ers` extension) and need only be specified if the data file has a different name. Some applications like CDROMS will always enforce the dot if there is no extension. You can have a header called `frog.ers` which contains the line `Datafile = "frog."`, and it will access the file correctly.
- **LastUpdated:** The date the file was last updated. For example,

```
LastUpdated = Mon Dec 2 03:14:55 GMT 1991
```
- **SensorName:** The location and name of the Camera File which has details of the sensor used to take the image. For example,

```
SensorName = "C:\Program Files\ERDAS\ERDAS ER Mapper 7.2\examples\Tutorial\camera2.cam"
```
- **SenseDate:** The date the data was collected. For example,

```
SenseDate = Mon Feb 5 06:39:03 GMT 1990
```

- **HeaderOffset:** Specifies in bytes the length of any header data in the data file, for non ER Mapper format BIL files. This means that, if a file is BIL format, with a fixed length header, specifying

```
HeaderOffset=512
```

in the ER Mapper header file will skip 512 bytes at the beginning of the data file. In this way, an ER Mapper header file can allow for any BIL format file with a fixed length header.

- **FFTInfo block:** In a transformed image, stores the original image information to be used to reconstruct the original image after a reverse transformation. See "[FFTInfo block](#)" below.

## Coordinate Space Block

The "*CoordinateSpace*" block specifies the datum and projection type, the type of coordinates used, the units and rotation from true North. For example,

```
CoordinateSpace Begin
Datum = "RAW"
Projection = "RAW"
CoordinateType = RAW
Units = "METERS"
Rotation = 0:0:0.0
CoordinateSpace End
```



*The Units entry does not apply to the cell-sizes shown in the CellInfo block.*

## Compulsory entries

- **Datum:** The datum for the map projection. Allowable values are RAW or one of the datums supported by ER Mapper (specified in quotes). For example,

```
Datum = "RAW" or Datum = "AGD66"
```

- **Projection:** The map projection for the image. Allowable values are RAW or one of the map projections supported by ER Mapper (specified in quotes). For example,

```
Projection = "RAW"
```

or

```
Projection = "TMAMG53"
```

- **CoordinateType:** Defines how coordinates are expressed. Note that quotes are not used in this entry. Allowed types are:
  - RAW —Coordinates expressed as meters in master coordinates.

- **EN** —Coordinates expressed as Easting Northing coordinate pair
- **LL** —Coordinates expressed as latitude and longitude in degrees.

For example,

```
CoordinateType = RAW
```

### Optional entries

- **Units:** The unit of length used in the image (and the RegistrationCoord block).

Allowed entries are:

- "natural"
- "METERS"
- "U.S. SURVEY FOOT"
- "IMPERIAL YARD"
- other units defined in the control file  
`'ERMAPPER\GDT_DATA\lenunits.dat'`

The most common units are "METERS" and "natural". If the units are not specified, they default to "METERS" for RAW images and "natural" for non-RAW images. Here, "natural" means the natural units for the projection. For example,

```
Units = "METERS"
```

- **Rotation:** Rotation defines the rotation of the image from true North in degrees counter-clockwise. For example,

```
Rotation = 0:0:0.0
```

defines no rotation from true North.

## RasterInfo Block

The RasterInfo block is a compulsory sub-block of the DatasetHeader block. It contains information about the image lines, bands and cells. The example below has both compulsory and optional entries.

```
RasterInfo Begin
  CellType = IEEE4ByteReal
  NullCellValue = -9999
  CellInfo Begin
    Xdimension = 50.0
    Ydimension = 50.0
  CellInfo End
  NrOfLines = 401
  NrOfCellsPerLine = 300
  RegistrationCoord Begin
    Eastings = 780810.6
    Northings = 7982367.8
  RegistrationCoord End
  RegistrationCellX = 0.0
  RegistrationCellY = 401.0
  NrOfBands = 1
  BandId Begin
    Value = "0.645"
    Width = 0.071
    Units = "um"
  BandId End
RasterInfo End
```

### Compulsory entries

- **CellType:** The type of data in the image. Allowed keywords are:
  - Unsigned8BitInteger
  - Signed8BitInteger
  - Unsigned16BitInteger
  - Signed16BitInteger
  - Signed32BitInteger
  - Unsigned32BitInteger
  - IEEE4ByteReal
  - IEEE8ByteReal

For example,

```
CellType = IEEE4ByteReal
```

- **NrOfLines:** Integer number of lines in the image. For example,

```
NrOfLines = 401
```

- **NrOfCellsPerLine:** Integer number of cells for each line in the image. For example,

```
NrOfCellsPerLine = 300
```

- **NrOfBands:** Integer number of bands in the image. For example,

```
NrOfBands = 3
```

## Optional entries

- **NullCellValue:** Indicates the value for "null cells". This is an optional indicator which may have any integer or real value. For example,

```
NullCellValue= -9999
```

This example specifies that any cell with the value "-9999" should be considered null data.

- **RegistrationCellX and RegistrationCellY :** The image X and Y coordinates of the cell which corresponds to the Registration Coordinate. Note that the `RegistrationCellX` and `RegistrationCellY` can be fractional values. If `RegistrationCellX` and `RegistrationCellY` are not specified, they are assumed to be (0,0), which is the top left corner of the image. For example

```
RegistrationCellX = 0.0  
RegistrationCellY = 401.0
```

- **CellInfo Block:** The X and Y dimensions of a data cell. See ["CellInfo block"](#) below.
- **RegistrationCoord Block:** The location of the point in the image specified by the `RegistrationCellX` and `RegistrationCellY` parameters. This block is required for non-RAW images. See ["RegistrationCoord block"](#) below.
- **BandID Block:** A description of what the bands in the image represent in physical terms. See ["RegistrationCoord block"](#) below.
- **SensorInfo Block:** Calibration details of the sensor (e.g. camera) used to take the image. Includes information on the location of Fiducial Points, lens focal length and location of the Principal Point. It is required for orthorectification. See ["SensorInfo block"](#) below.
- **WarpControl Block:** Stores information about rectification including the ground control points. See ["WarpControl block"](#) below.
- **RegionInfo Block:** Defines regions in the image and lists statistics for them. See ["RegionInfo block"](#) below.

## CellInfo Block

This optional block is a sub-block of the RasterInfo Block. It specifies the X and Y dimensions of an image cell.

```
CellInfo Begin
  Xdimension = 50.0
  Ydimension = 50.0
CellInfo End
```

- **Xdimension:** X dimension of each image cell in meters. Default is 1.

```
Xdimension = 50.0
```

- **Ydimension:** Y dimension of each image cell in meters. Default is 1.

```
Ydimension = 50.0
```

## Registration Coord Block

The "RegistrationCoord" block is a sub-block of the RasterInfo block. It defines the location of the registration cell in the image specified by the RegistrationCellX and RegistrationCellY parameters. The coordinates must be compatible with the system specified in the CoordinateType parameter. Eastings/Northings or MetersX/MetersY or Latitude/Longitude are specified. The block is required for non-RAW images. For example:

```
RegistrationCoord Begin
  Eastings= 780810.6
  Northings= 7982367.8
RegistrationCoord End
```

```
Eastings = 780810.6
Northings = 7982367.8
```

or

```
MetersX = 8625.0
MetersY = 9225.0
```

or

```
Latitude = 34:16:54.98
Longitude = 140:34:3.2
```



## BandId Block

The optional "BandId" block is a sub-block of the RasterInfo block. It specifies what each of the bands in the image represents in physical terms. The "Width" value is optional and specifies the spectral width of the image band in the units defined by the Units parameter. For example,

```
BandId Begin
  Value = "0.645"
  Width = 0.071
  Units = "um"
BandId End
```

### Compulsory entries

- **Value:** Value description to appear in the band button in the Algorithm window

```
Value = "0.645"
```

- **Units:** Units for band description

```
Units = "um"
```

### Optional entries

- **Width:** Spectral width of the image band, in the units defined below, used for closest spectral matching with bands from other images

```
Width = 0.071
```

## SensorInfo Block

The optional "SensorInfo" block is a sub-block of the RasterInfo block. It provides details of the sensor (e.g. camera) used to take the image.

This information is used to orthorectify the image.

For example,:

```
SensorInfo Begin
  CameraManufacturer= "Wild"
  CameraModel = "RC20"
  LensSerialNr = "13115"
  CalibrationDate = Wed Oct 23 00:00:00 GMT 1996
  SensorType = MetricCamera
  PlatformType = Aerial
  FiducialInfo Begin
    PrinciplePointOffsetX = 0
    PrinciplePointOffsetY = 0
    FiducialPointTopLeft Begin
      IsOn = Yes
      IsLocked = No
      CellX = 210.4633887599771
      CellY = 320.1004736175112
      OffsetX = -105.987
      OffsetY = 106.007
```

```

FiducialPointTopLeft End
FiducialPointTopRight Begin
  IsOn = Yes
  IsLocked = No
  CellX = 5220.712372586905
  CellY = 321.5464165158579
  OffsetX = 106.008
  OffsetY = 106.016
FiducialPointTopRight End
FiducialPointBottomLeft Begin
  IsOn = Yes
  IsLocked = No
  CellX = 202.7989504537253
  CellY = 5332.408872554061
  OffsetX = -105.986
  OffsetY = -105.99
FiducialPointBottomLeft End
FiducialPointBottomRight Begin
  IsOn = Yes
  IsLocked = No
  CellX = 5213.025106596184
  CellY = 5331.571326150155
  OffsetX = 106.01
  OffsetY = -105.99
FiducialPointBottomRight End
FiducialInfo End
FocalLength = 152.793
PrinciplePointX = 1
PrinciplePointY = 1
SensorInfo End

```

### Compulsory entries

- **SensorType:** The type of sensor used. Value can be MetricCamera, nonMetricCamera or LineArray.

```
SensorType = MetricCamera
```

- **PlatformType:** The type of platform on which the sensor was mounted. Value can be Aerial, Terrestrial or Satellite.

```
PlatformType = Aerial
```

- **FiducialInfo Block:** Contains information on the number and location of Fiducial Points.

### Optional entries

- **CameraManufacturer:** The name of the camera manufacturer, for example:

```
CameraManufacturer = "Leica"
```

- **CameraModel:** The model of camera used, for example:

```
CameraModel = "RC20"
```

- **LensSerialNr:** The serial number of the lens, for example:

```
LensSerialNr = "100100"
```

- **CalibrationDate:** The date of the camera calibration report from which this information is obtained.

```
CalibrationDate = Mon Jun 01 00:00:00 GMT 1998
```

## FiducialInfo Block

The "FiducialInfo" block is a sub-block of the SensorInfo block. It provides details of the Fiducial Points inserted by the camera.

- **PrincipalPointOffsetX,Y:** The amount, in millimeters, that the Principal Point is offset from the lens center due to lens distortion.

```
PrinciplePointOffsetX = 0.006  
PrinciplePointOffsetY = 0
```

## Fiducial point block

Each Fiducial Point has one of these blocks, which defines its position and status. For example:

```
FiducialPointMiddleTop Begin  
  IsOn = Yes  
  IsLocked = No  
  CellX = 2716.929167534793  
  CellY = 227.1583440160684  
  OffsetX = 0.011  
  OffsetY = 110.007  
FiducialPointMiddleTop End
```

- **IsOn:** Flag indicating whether the Fiducial Point is used in calculating RMS errors.
- **Islocked:** Flag indicating whether the Fiducial Point position is locked for further adjusting.
- **CellX, CellY:** Cell position of the Fiducial Point in the image.
- **OffsetX, OffsetY:** Fiducial point position relative to the Principal Point.

## RegionInfo Block

The optional 'RegionInfo' block is a sub-block of the RasterInfo block. It defines regions in the image and lists statistics for those regions. Where a number of regions have been defined for a multi-band image this block may be of a considerable size. For import and export purposes this block can be ignored.

- **Type:** Must be "Polygon". Other types may be generated by ER Mapper but may be ignored.

```
Value = "0.645"
```

- **RegionName:** Name for the region to be used on menus.

```
Width = 0.071
```

- **SubRegion:** Coordinates of points on the region outline, in image cell coordinates.

```
SubRegion = {  
    33.6551 55.6845  
    219.3385 68.4455  
}
```

- **Stats Block:** Ignore this block. Generated by ER Mapper.

## WarpControl Block

The optional 'WarpControl' block is a sub-block of the RasterInfo block. It defines the parameters to be used for the image to be rectified. This information does not pertain to the image referenced by this header file, but to the resultant rectified image.

An example WarpControl Block is as follows:

```
WarpControl Begin  
    WarpType = Ortho  
    WarpSampling = Nearest  
    Rotation = 0  
    DemFile = "San_Diego_DEM.ers"  
    DemBandNr = 1  
    ChooseGcpsFromDigitizer = No  
    ChooseGcpsFromImage = Yes  
    GcpsChosenFrom = "San_Diego_rectified.alg"  
    OutputFile = "San_Diego_Airphoto_34_rectified.ers"  
    UseAverageHeight = No  
    AverageHeight = 0  
    OutputCellSizeX = 0.8784871184780805  
    OutputCellSizeY = 0.8784871184780805  
    OutputHasNullCells = Yes  
    OutputNullCellValue = 0  
    Correction Begin  
        RadialLens = No  
        PolyLens = No  
        Atmospheric = No  
        EarthCurvature = No  
    Correction End  
    GivenOrthoInfo Begin  
        AttitudeOmega = 0  
        AttitudePhi = 0  
        AttitudeKappa = 0  
        ExposureCenterX = 0  
        ExposureCenterY = 0
```

```

ExposureCenterZ = 0
SCALE = 0
CoordinateSpace Begin
    Datum = "RAW"
    Projection = "RAW"
    CoordinateType = RAW
    Rotation = 0:0:0.0
CoordinateSpace End
GivenOrthoInfo End
CoordinateSpace Begin
    Datum = "NAD27"
    Projection = "NUTM11"
    CoordinateType = EN
    Units = "METERS"
    Rotation = 0:0:0.0
CoordinateSpace End
Extents Begin
    TopLeftCorner Begin
        Eastings = 481254.9675890448
        Northings = 3623804.501505829
    TopLeftCorner End
    BottomRightCorner Begin
        Eastings = 485756.9891375387
        Northings = 3619284.200966488
    BottomRightCorner End
Extents End
ControlPoints = {
    "1035" Yes No 2344.650885 3546.419458 483270.73
3620906.21 3.105
    "165" Yes No 753.008933 3870.075769 481853.04
3620609.47 3.956
    "1033" No No 1135.630546 3300.182792 482174.61
3621108.2 3.781
    "166" Yes No 1899.884085 1809.975198 482865.1
3622446.77 9.336
    "1048" Yes No 4543.843305 2126.060086 485146.6
3622136.67 89.52
    "1047" Yes No 4023.228765 3093.897173 484710.74
3621312.38 69.78
    "1049" Yes No 4285.065234 1075.63221 484932.5
3623042.13 90.60
    "1050" No No 4421.36311 195.759133 485056.3
3623816.47 90.47
    "1043" Yes No 3328.0604 456.3793 484120.33
3623601.21 81.31
    "1040" Yes No 2446.265469 586.068957 483353.38
3623498.02 83.91
    "1036" Yes No 628.09969 1281.942016 481698.47
3622934.34 4.019
}

```

WarpControl End

- **WarpType:** Specifies the type of geocoding to be done on the image. Allowable values are:
  - Ortho
  - OrthoAdvanced
  - Polynomial
  - Triangulation
- **Rotation:** Specifies the amount by which the image is to be rotated anti-clockwise in decimal degrees. 0 indicates no rotation.
- **ChooseGcpsFromDigitizer:** Specifies whether a digitizer was used to select Ground Control Points (GCPs) used for geocoding the image. Value can be either 'Yes' or 'No'.
- **ChooseGcpsFromImage:** Specifies whether a geocoded (corrected) image was used to select Ground Control Points (GCPs) to be used for rectifying the image. Value can be either 'Yes' or 'No'.
- **UseAverageHeight:** Specifies whether an average height is to be used instead of a DEM for orthorectifying the image. Value can be either 'Yes' or 'No'.
- **AverageHeight:** Specifies the average height to be used if the UseAverageHeight was set to 'Yes'.
- **Correction Block:** Specifies the type of correction to be done when the image is geocoded.
- **GivenOrthoInfo Block:** Contains the parameters to be used for orthorectifying the image. This is only applicable if WarpType is set to "OrthoAdvanced".
- **CoordinateSpace Block:** Specifies the new coordinate type and rotation of the image.
- **ControlPoints:** Specifies the location and properties of Ground Control Points (GCPs).

The parameters in the ControlPoint entries are:

1. Name: The name of the point. For example, "Name"
2. On: Whether the point is taken into account when calculating the rectification coefficients.

3. Editable: Whether the point can be edited or not. Allowable values are Yes or No.
4. FROM cellx: The x coordinate of the point in the FROM coordinate system.
5. FROM celly: The y coordinate of the point in the FROM coordinate system.
6. TO x: The x coordinate of the point in the TO coordinate system.
7. TO y: The y coordinate of the point in the TO coordinate system.
8. TO z: The z coordinate of the point in the TO coordinate system. This is only required if WarpType is set to "Ortho".

## Correction Block

The Correction Block is a sub-block of the WarpControl block. It indicates the types of correction to be carried out on the image during geocoding. This is currently not used by ER Mapper.

```
Correction Begin
RadialLens = No
PolyLens = No
Atmospheric = No
EarthCurvature = No
Correction End
```

- **RadialLens:** Radial lens distortion is introduced by irregularities in the camera (sensor) lens.
- **PolyLens:** Poly lens distortion.
- **Atmospheric:** Atmospheric distortion is caused by changes in the earth's atmosphere.
- **EarthCurvature:** Distortion caused by the earth's curvature.

## GivenOrthoInfo Block

The GivenOrthoInfo block is a sub-block of the WarpControl block. It contains information on the exact position of the sensor platform when the image was taken. This is known as exterior orientation and is used to orthorectify the image.

```
GivenOrthoInfo Begin
  AttitudeOmega = 0
  AttitudePhi = 0
  AttitudeKappa = 0
  ExposureCenterX = 0
  ExposureCenterY = 0
  ExposureCenterZ = 0
  SCALE = 0
  CoordinateSpace Begin
    CoordinateType = None
    Rotation = 0:0:0.0
  CoordinateSpace End
GivenOrthoInfo End
```

- **AttitudeOmega:** The pitch about the Y axis, in radians.
- **AttitudePhi:** The roll about the X axis, in radians.
- **AttitudeKappa:** The yaw about the Z axis, in radians.
- **ExposureCenterX:** The position of the exposure center of the camera in the X plane of the image. This is in the same units as defined in the CoordinateSpace block.
- **ExposureCenterY:** The position of the exposure center of the camera in the Y plane of the image. This is in the same units as defined in the CoordinateSpace block.
- **ExposureCenterZ:** The height of the exposure center of the camera. This is in the same units as defined in the CoordinateSpace block.
- **SCALE:** The scale of the image as a decimal value. This parameter is calculated internally by ER Mapper.



## FFTInfo Block

The optional 'FFTInfo' block is a sub-block of the DatasetHeader block. When you carry out a Fourier transform the information about the original image is stored here. This is used to reconstruct the original image when reversing the transformation. For example,

```
FFTInfo Begin
  ForwardTransform = FFT
  SpectrumAmount = Full
  SpatialDataset = "Shared_Data/FFT_Input_Image.ers"
  PadCellX = 10
  PadCellY = 10
  CoordinateSpace Begin
    Datum = "NAD27"
    Projection = "NUTM11"
    CoordinateType = EN
    Rotation = 0:0:0.0
  CoordinateSpace End
  RasterInfo Begin

.... the RasterInfo block for the original image ....

  RegionInfo End
  RasterInfo End
FFTInfo End
```

## Example dataset header files

### Example Landsat header file

```
DatasetHeader Begin
  Version = "3.0"
  LastUpdated = Tue Nov 19 02:16:39 GMT 1991
  SensorName = "Landsat_TM"
  SenseDate = Fri May 18 07:08:46 GMT 1990
  DataSetType = ERStorage
  DataType = Raster
  ByteOrder = MSBFirst
  CoordinateSpace Begin
    Datum = "RAW"
    Projection = "RAW"
    CoordinateType = RAW
    Rotation = 0:0:0.0
  CoordinateSpace End
  RasterInfo Begin
    CellType = Unsigned8BitInteger
    NullCellValue = 0
    CellInfo Begin
      Xdimension = 30
      Ydimension = 30
    CellInfo End
    NrOfLines = 1000
    NrOfCellsPerLine = 1000
    NrOfBands = 7
  RasterInfo End
DatasetHeader End
```

## Georeferenced dataset header file

Here is a dataset header file for an image which is georeferenced to AMG coordinates.

```
DatasetHeader Begin
  DataSetType = ERStorage
  HeaderOffset= 512
  DataType = Raster
  ByteOrder      = MSBFirst
  CoordinateSpace Begin
    Datum = "AGD66"
    Projection = "TMAMG53"
    CoordinateType = EN
    Rotation = 0:0:0.0
  CoordinateSpace End
  RasterInfo Begin
    CellType = IEEE4ByteReal
    NullCellValue = -9999
    CellInfo Begin
      Xdimension = 50.0
      Ydimension = 50.0
    CellInfo End
    NrOfLines = 401
    NrOfCellsPerLine = 300
    RegistrationCoord Begin
      Eastings = 780810.6
      Northings = 7982367.8
    RegistrationCoord End
    RegistrationCellX = 0.0
    RegistrationCellY = 401.0
    NrOfBands = 1
    BandId Begin
      Value = "0.645"
      Width = 0.071
      Units = "um"
    BandId End
  RasterInfo End
DatasetHeader End
```

## Example of a complex dataset header file

This example is a more complex header with many of the available options. It includes orthorectification and region information which has been added to the header by ER Mapper as it calculates statistical information.

```
DatasetHeader Begin
  Version = "5.7"
  LastUpdated = Mon Aug 10 03:43:43 GMT 1998
  SensorName n=
  "C:\Program Files\ERDAS\ERDAS ER Mapper 7.2\
  examples\Tutorial\camera2.cam"
  DataSetType = ERStorage
```

```

DataType = Raster
ByteOrder = MSBFirst
CoordinateSpace Begin
    Datum = "NAD27"
    Projection = "NUTM11"
    CoordinateType = EN
    Rotation = 0:0:0.0
CoordinateSpace End
RasterInfo Begin
    CellType = Unsigned8BitInteger
    CellInfo Begin
        Xdimension = 0.75
        Ydimension = 0.75
    CellInfo End
    NrOfLines = 694
    NrOfCellsPerLine = 652
    RegistrationCoord Begin
        Eastings = 484875.8624313
        Northings = 3620515.084881
    RegistrationCoord End
    NrOfBands = 3
    BandId Begin
        Value = "Red"
    BandId End
    BandId Begin
        Value = "Green"
    BandId End
    BandId Begin
        Value = "Blue"
    BandId End
    SensorInfo Begin
        CameraManufacturer = "Leica"
        CameraModel = "RC20"
        LensSerialNr = "100100"
        CalibrationDate = Mon Jun 01 00:00:00 GMT 1970
        SensorType = MetricCamera
        PlatformType = Aerial
        FiducialInfo Begin
            PrinciplePointOffsetX = 0.006
            PrinciplePointOffsetY = 0
            FiducialPointTopLeft Begin
                IsOn = Yes
                IsLocked = No
                CellX = 210.4633887599771
                CellY = 320.1004736175112
                OffsetX = -105.987
                OffsetY = 106.007
            FiducialPointTopLeft End
            FiducialPointTopRight Begin
                IsOn = Yes

```

```

IsLocked = No
CellX = 5220.712372586905
CellY = 321.5464165158579
OffsetX = 106.008
OffsetY = 106.016
FiducialPointTopRight End
FiducialPointBottomLeft Begin
  IsOn = Yes
  IsLocked = No
  CellX = 202.7989504537253
  CellY = 5332.408872554061
  OffsetX = -105.986
  OffsetY = -105.99
FiducialPointBottomLeft End
FiducialPointBottomRight Begin
  IsOn = Yes
  IsLocked = No
  CellX = 5213.025106596184
  CellY = 5331.571326150155
  OffsetX = 106.01
  OffsetY = -105.99
FiducialPointBottomRight End
FiducialPointMiddleLeft Begin
  IsOn = Yes
  IsLocked = No
  CellX = 112.1652994450148
  CellY = 2828.169317438489
  OffsetX = -109.989
  OffsetY = 0.012
FiducialPointMiddleLeft End
FiducialPointMiddleRight Begin
  IsOn = Yes
  IsLocked = No
  CellX = 5310.975805982356
  CellY = 2828.913786847189
  OffsetX = 110.01
  OffsetY = 0.013
FiducialPointMiddleRight End
FiducialPointMiddleTop Begin
  IsOn = Yes
  IsLocked = No
  CellX = 2716.929167534793
  CellY = 227.1583440160684
  OffsetX = 0.011
  OffsetY = 110.007
FiducialPointMiddleTop End
FiducialPointMiddleBottom Begin
  IsOn = Yes
  IsLocked = No
  CellX = 2708.487218792691

```

```

        CellY = 5428.231114609829
        OffsetX = 0.005
        OffsetY = -109.991
        FiducialPointMiddleBottom End
    FiducialInfo End
    FocalLength = 152.793
    PrinciplePointX = 1
    PrinciplePointY = 1
SensorInfo End
WarpControl Begin
    WarpType = Ortho
    WarpSampling = Nearest
    Rotation = 0
    ChooseGcpsFromDigitizer = No
    ChooseGcpsFromImage = Yes
    GcpsChosenFrom =
        "C:\Program Files\ERDAS\ERDAS ER Mapper
7.2\examples\Data_Types\Airphoto\RGB.alg"
    UseAverageHeight = Yes
    AverageHeight = 50
Correction Begin
    RadialLens = No
    PolyLens = No
    Atmospheric = No
    EarthCurvature = No
Correction End
GivenOrthoInfo Begin
    AttitudeOmega = 0
    AttitudePhi = 0
    AttitudeKappa = 0
    ExposureCenterX = 0
    ExposureCenterY = 0
    ExposureCenterZ = 0
    SCALE = 0
    CoordinateSpace Begin
        CoordinateType = None
        Rotation = 0:0:0.0
    CoordinateSpace End
GivenOrthoInfo End
CoordinateSpace Begin
    Datum = "NAD27"
    Projection = "NUTM11"
    CoordinateType = EN
    Rotation = 0:0:0.0
CoordinateSpace End
ControlPoints = {
    "1035" Yes No 2344.650885 3546.419458 483270.73
3620906.21 3.105
    "165" Yes No 1135.630546 3300.182792 482174.61
3621108.2 3.781

```

```

    "166" Yes No 1899.884085 1809.975198 482865.1
    3622446.77 9.336
    "1048" Yes No 4543.843305 2126.060086 485146.6
    3622136.67 89.522
    "1047" Yes No 4023.228765 3093.897173 484710.74
    3621312.38 69.787
    "1049" Yes No 4285.065234 1075.63221 484932.5
    3623042.13 90.606
    "1050" No No 4421.36311 195.759133 485056.3
    3623816.47 90.474
  }
  WarpControl End
  RegionInfo Begin
    Type = Polygon
    RegionName = "All"
    RGBcolour Begin
      Red = 0
      Green = 63057
      Blue = 56957
    RGBcolour End
    SubRegion= {
      0 0
      0 694
      652 694
      652 0
    }
  Stats Begin
    SubsampleRate = 4
    NumberOfBands = 3
    NumberOfNonNullCells = {
      28362 28362 28362
    }
    MinimumValue= {
      0 0 0
    }
    MaximumValue= {
      247 246 247
    }
    MeanValue= {
      110.6474860729 106.2357379592
      96.27251251675
    }
    MedianValue= {
      105 94 90
    }
    CovarianceMatrix= {
      8092.874707019 6894.430764364
      5603.49383238
      6894.430764364 6346.305766934
      5192.448925684

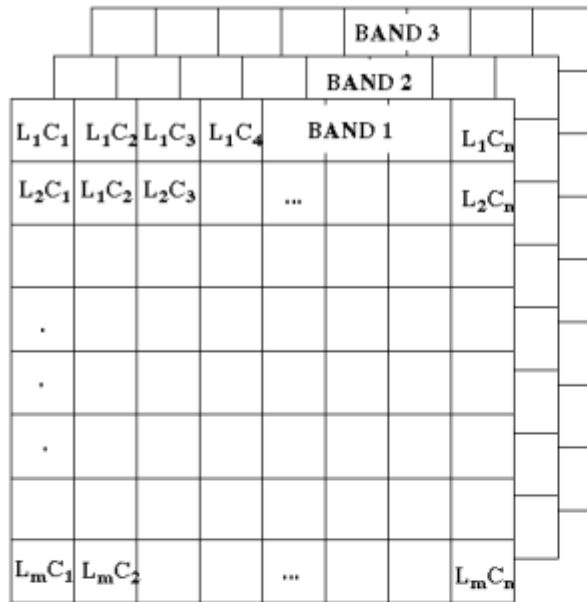
```

```
                    5603.49383238  5192.448925684
4400.993009794
                    }
                    Stats End
                    RegionInfo End
                    RasterInfo End
DatasetHeader End
```

### **Raster data file**

An ER Mapper raster data file contains binary data in Band-Interleaved-by Line (BIL) format. The type and amount of data is specified in the header file. The figure below illustrates the ordering of data in an ER Mapper raster data file.





A multi-band image with  $m$  lines and  $n$  cells per line

The RasterInfo block parameters for this example are:

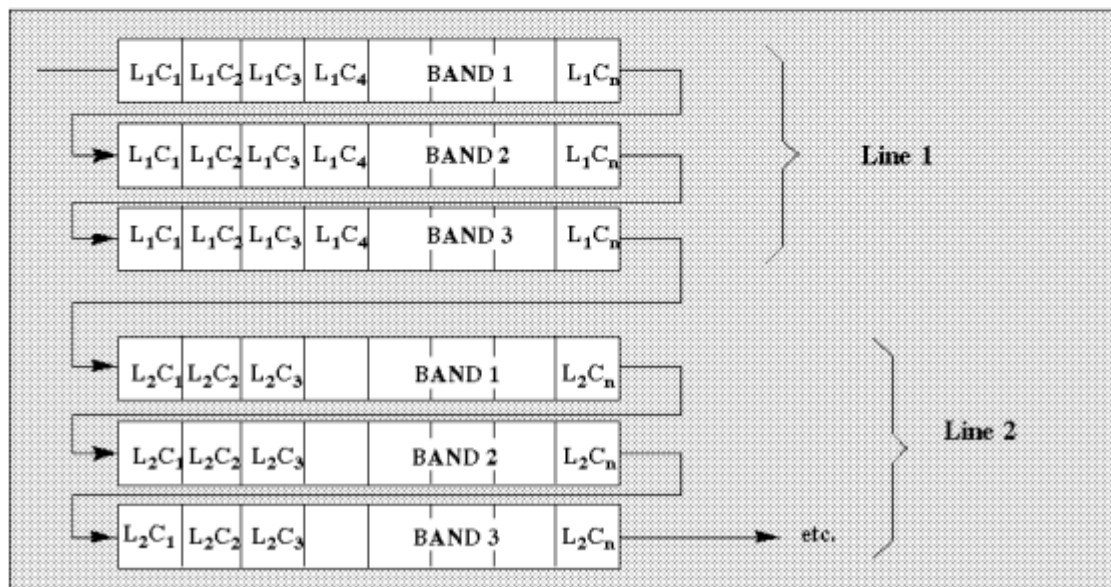
NrOfLines =  $m$   
 NrOfCellsPerLine =  $n$   
 NrOfBands = 3

Other relevant RasterInfo block parameters are, for example:

CellType = Signed8BitInteger

Other relevant DatasetHeader block parameters are, for example:

ByteOrder = MSBFirst





# Vector Datasets and Header Files (.erv)

## Introduction

An ER Mapper Vector dataset is made up of two files:

- the dataset header file, and
- the data file.

## The header file

The dataset header file is an ASCII file describing the vector data in the data file. The dataset header file generally has the same name as the data file that it is describing, with the file extension `“.erv”` added. For example, `“Loxton_contour.erv”` is the dataset header file name corresponding to the vector data file `“Loxton_contour”`. The header file may point to a data file that has a different name via the `DataFile` field in the header file.

## The data file

The vector data file contains the data itself, also in ASCII format. The data file name has no extension. For example, `“Loxton_contour”` is the vector data file described by the `“Loxton_contour.erv”` Header file.

This chapter details the formats for these two files.

## Vector header file format

The vector file format is similar to the raster header file format, except that the VectorInfo block takes the place of the RasterInfo block. An example vector header file is listed below, followed by a description of the VectorInfo block entries.

```
DatasetHeader Begin
  Version = "5.0"
  DataFile = "Shared_Data/Australia"
  LastUpdated = Fri Nov 15 02:49:12 GMT 1991
  SenseDate = Tue Sep 4 04:56:37 GMT 1990
  DataSetType = ERStorage
  DataType = Vector
  ByteOrder = MSBFirst
  CoordinateSpace Begin
    Datum = "AGD66"
    Projection = "TMAMG54"
    CoordinateType = EN
    Rotation = 0:0:0.0
  CoordinateSpace End
  VectorInfo Begin
    Type = ERVEC
    FileFormat = ASCII
    Extents Begin
      TopLeftCorner Begin
        Eastings = 453982
        Northings = 6210075
      TopLeftCorner End
      BottomRightCorner Begin
        Eastings = 477048
        Northings = 6182287
      BottomRightCorner End
    Extents End
  VectorInfo End
DatasetHeader End
```

The entire header file is contained in the DatasetHeader Block. This holds information about the data source and has two sub-blocks, the CoordinateSpace Block and the VectorInfo Block.

- The CoordinateSpace Block defines the dataset map projection.
- The VectorInfo Block defines the characteristics of the data in the accompanying data file and the file extents.

The next section describes the entries in the DatasetHeader block and the subsequent sections describe sub-blocks.

## DatasetHeader Block

This block contains the whole of the Header file. It has two compulsory sub-blocks: the CoordinateSpace block and the VectorInfo block. The DatasetHeader entries and the CoordinateSpace block entries are the same as for raster dataset headers. They are repeated here for completeness.

A simple vector header file is shown below.

```
DatasetHeader Begin
  Version          = "5.5"
  DataFile         = "Shared_Data/Australia"
  LastUpdated      = Mon Dec 2 03:14:55 GMT 1994
  DataSetType      = ERStorage
  DataType         = Vector
  ByteOrder        = MSBFirst
  CoordinateSpace Begin
    Datum           = "RAW"
    Projection       = "RAW"
    CoordinateType  = RAW
  CoordinateSpace End
  VectorInfo Begin
    Type            = ERVEC
    FileFormat      = ASCII
  Extents Begin
    TopLeftCorner Begin
      MetersX       = 0
      MetersY       = 0
    TopLeftCorner End
    BottomRightCorner Begin
      MetersX       = 1000
      MetersY       = 1000
    BottomRightCorner End
  Extents End
  VectorInfo End
DatasetHeader End
```

The entries of a vector dataset header are described below:

### Compulsory entries

- **Version:** Used to define the version. For example,

```
Version = "5.5"
```

- **DataSetType:** The type of dataset to be displayed. Currently the only allowed value is ERStorage.

```
DataSetType = ERStorage
```

There are many import utilities that may be used to convert datasets from a foreign file format into ER Storage format. See Chapter 23. "Importing using a command line".

- **DataType:** `DataType` tells ER Mapper what type of dataset it is. Values allowed are `Raster` or `Vector`. For example,

```
DataType = Vector
```




---

*Raster dataset entries are detailed in Chapter 6, "Raster datasets and header files (.ers)".*

- **ByteOrder:** Indicates the byte ordering of the data. Can be either:
  - `MSBFirst` - Most significant byte first.
  - `LSBFirst` - Least significant byte first.
- For example,

```
ByteOrder = MSBFirst
```




---

*Byte order is not significant for ASCII vector files.*

- **CoordinateSpace block:** Defines the dataset projection. See "*CoordinateSpace block*" below.
- **VectorInfo block:** Defines the dataset format. See "*VectorInfo block*" below.
- **DataFile:** The name of the data file described by this header file. The default is the same name (without the `.erv` extension) and need only be specified if the data file has a different name. Some applications like CDROMS will always enforce the dot if there is no extension. You can have a header called '`frog.erv`' which contains the line '`Datafile = "frog."`', and it will access the file correctly.
- **LastUpdated:** The date the file was last updated. For example,

```
LastUpdated = Mon Dec 2 03:14:55 GMT 1991
```

- **SenseDate:** The date the data was collected. For example,

```
SenseDate = Mon Feb 5 06:39:03 GMT 1990
```

## Optional entries

## CoordinateSpace Block

The "CoordinateSpace" block specifies the datum and projection type, the type of coordinates used, the units and rotation from true North. For example,

```
CoordinateSpace Begin
Datum = "RAW"
Projection = "RAW"
CoordinateType = RAW
Units = "METERS"
Rotation = 0:0:0.0
CoordinateSpace End
```

### Compulsory entries

- **Datum:** The datum for the map projection. Allowable values are RAW or one of the datums supported by ER Mapper (specified in quotes). For example,

```
Datum = "RAW" or Datum = "AGD66"
```



---

*See Appendix D "Supported map projections and datums" in the ER Mapper User Guide for a list of available datums.*

- **Projection:** The map projection for the dataset. Allowable values are RAW or one of the map projections supported by ER Mapper (specified in quotes). For example,

```
Projection = "RAW"
or
Projection = "TMAMG53"
```



---

*See Appendix D "Supported map projections and datums" in the ER Mapper User Guide for a list of available map projections.*

- **CoordinateType:** Defines how coordinates are expressed. Note that quotes are not used in this entry. Allowed types are:
  - RAW —Coordinates expressed as metres in master coordinates.
  - EN —Coordinates expressed as Easting Northing coordinate pair
  - LATLONG —Coordinates expressed as latitude and longitude in degrees.

For example,

```
CoordinateType = RAW
```

## Optional entries

- **Units:** The unit of length used in the dataset (and the RegistrationCoord block).

Allowed entries are:

- "natural"
- "METERS"
- "U.S. SURVEY FOOT"
- "IMPERIAL YARD"
- other units defined in the control file  
**ERMAPPER\GDT\_DATA\lenunits.dat.**

The most common units are "METERS" and "natural". If the units are not specified, they default to "METERS" for RAW datasets and "natural" for non-RAW datasets. Here, "natural" means the natural units for the projection. For example,

```
Units = "METERS"
```

- **Rotation:** Rotation defines the rotation of the dataset from true North in degrees counter-clockwise. For example,

```
Rotation = 0:0:0.0
```

defines no rotation from true North.

## VectorInfo Block

The VectorInfo block contains information about the type of data in the data file. All the entries are required.

- **Type:** The type of data in the dataset. Allowed keywords are:
  - ERVEC
  - ERS (for backward compatibility only)

For example,

```
CellType = ERVEC
```

- **FileFormat:** Only ASCII allowed in this release. For "ERS", "ASCII" and "BINARY" are allowed.

```
FileFormat = ASCII
```

- **Extents Block:** Defines the vector data extents. See "Extents block" below.



## Extents Block

The Extents block defines the extents of the vector data by specifying the coordinates of the top left and bottom right corners of the dataset. It is made up of two sub-blocks. These are shown in the example below,

```
Extents Begin
  TopLeftCorner Begin
    Eastings = 780810.6
    Northings = 7982367.8
  TopLeftCorner End
  BottomRightCorner Begin
    Eastings = 780810.6
    Northings = 7982367.8
  BottomRightCorner End
Extents End
```

The coordinates can be in Eastings/Northings or MetersX/MetersY or Latitude/Longitude, compatible with the CoordinateType parameter. For example,

```
MetersX = 8625.0
MetersY = 9225.0
```

or

```
Latitude = 34:16:54.98
Longitude = 140:34:3.2
```

## Example vector header files

### Example "RAW" vector dataset header file

The dataset header below is designed to overlay a 1000x1000 cell "raw" raster dataset. Example georeferenced vector dataset header "**Newcastle\_detail.erv**"

## Example of a dataset header for a georeferenced vector dataset.

```
DatasetHeader Begin
  Version          = "3.0"
  LastUpdated      = Thu Nov 28 05:42:01 GMT 1991
  SenseDate        = Thu May 31 21:46:26 GMT 1990
  DataSetType      = ERStorage
  DataType         = Vector
  ByteOrder        = MSBFirst
  CoordinateSpace Begin
    Datum           = "AGD66"
    Projection       = "TMAMG54"
    CoordinateType  = EN
  CoordinateSpace End
  VectorInfo Begin
    Type            = ERVEC
    FileFormat      = ASCII
  Extents Begin
    TopLeftCorner Begin
      Eastings      = 778062.95327
      Northings     = 7997191.179595
    TopLeftCorner End
    BottomRightCorner Begin
      Eastings      = 827209.421794
      Northings     = 7930501.866287
    BottomRightCorner End
  Extents End
  VectorInfo End
DatasetHeader End
```

### Vector data file

Below is a simple vector data file containing a list of object specifications. Note the following points:

- ER Mapper vector data files list object data specifications in ASCII format.
- In general there is one object per line
- Object specifications end with a full-stop (period) "."
- Consecutive fields in an object specification are delimited by a comma, but there is no comma after the last field.
- Real numbers may or may not contain a decimal point (that is, may be reals or integers), but for efficiency reasons exponent form may not be used.
- ER Mapper vector data processing is not space sensitive - spaces are simply ignored. Thus the indentation in text object specifications is optional.

- ER Mapper is sensitive to newline characters. Specifications for objects of type point, box, map\_box and oval must only take up one line each. In text object specifications, each line of literal text begins on a new line. Polyline, polygon and map\_polygon objects are not necessarily single line. Newline characters are inserted after every 20 pairs of coordinates.

```

box(,111.437,726.765,222.055,810.293,0,1,0,255,0,0,0) .
oval(,120.467,568.74,190.45,640.98,0,1,0,20,63,25,0) .
polygon(,5,[319.127,196.253,355.247,412.973,325.9,548.423,321.385,647.753,319.1
27,582.285],0,1,0,0,-1,-1,-1,0) .
poly(,2,[319.127,528.105,219.797,537.135],0,1,0,0,0,0,253,42,234,0) .
poly(,6,[0.5149998,0.5865908,0.4599999,0.5923862,0.5499998,0.658068,0.5899998,0
.5305681,0.4824998,0.3779545,0.3774999,0.3586363],2,4.5,9,0,0,0,253,42,234,1) .
point(a point,0.5474999,0.7488636,0,25,13,226,1) .
map_box("Name =
\Scale_Bar/Tick"\n",0.65749,0.34704,0.8249999,0.4011363,0,1,0,-1,-1,-
1,0,1) .
map_polygon("Name = \Clip_Mask/Inside"\n\n"Color" = \253,0,9"\n"Border
Color" =
\254,254,254""",4,[0.7149999,0.6310227,0.6874999,0.6503409,0.8549999,
0.6657954,0.8474999,0.5807954],0,1,0,0,-1,-1,-1,0,1) .
vtext(,0.0575,0.4359091,0.0575,41.0659101,877.6655231,0.4359091,Courier,0,40.63
,1,0,0,0,253,13,22,1,[
    "this is variable text, page relative"])).

```

The syntax for the object types in ER Mapper vector data files is described below. It is important to note that the syntax is line end sensitive but not space sensitive. The italicized specifiers are described after t\e object syntax.

### point

Syntax: point (attribute, x, y, spare, r, g, b, page).

For example:

```
point ("a point", 0.5474999,0.7488636,0,25,13,226,1)
```

### box

Syntax: box (attribute, ltx, lty, rbx, rby, fill, width, pen, r, g, b, page).

For example:

```
box ("a box", 58236.3500.56, 59236.35, 62600.56, 0, 0,
1, 255, 0, 0, 0)
```

### oval

Syntax: oval (attribute, ltx, lty, rbx, rby, fill, width, pen, r, g, b, page).

For example:

```
oval ("a circle", 250, 150, 300, 100, 3, 0, 0, -1, -1 -  
1, 0)
```

### **fixed text**

Syntax: text (attribute, x, y, font, style, size, nlines, just, angle, pen, r, g, b, page, [  
"line\_of\_text",

"line\_of\_text"]).

For example:

```
text ("A line of text", 5836.35, 6100.56, Helvetica-  
Bold, 1, 12, 1, 0, 0, 1, 0, 0, 255, 1, [  
"This is fixed text."])
```

### **variable text**

Syntax: vtext(attribute, x, y, ltx, lty, rbx, rby, font, style, size, nlines, just, angle, pen, r, g, b, page, [  
"line\_of\_text",

"line\_of\_text"]).

For example:

```
vtext(, 0.0575, 0.4359091, 0.0575, 41.0659101, 877.6655231,  
0.4359091, courier, 0, 40.63, 1, 0, 0, 0, 253, 13, 22, 1, [  
"this is variable text, page relative"])
```

### **polyline**

Syntax: poly (attribute, npoints, [X1, Y1, . . . Xn, Yn], end, width, pen, reserved, curved, fill, r, g, b, page)

For example:

```
poly (, 2, [0.5, 0.5, 2, 2], 2, 3, 1, 0, 0, 1, 220, 0,  
220, 1)
```

### **polygon**

Syntax: polygon (attribute, npoints, [X1, Y1, . . . Xn, Yn], fill, width, pen, curved, r, g, b, page)

For example:

```
polygon(, 5, [319.127, 196.253, 355.247, 412.973, 325.9, 548.  
423, 321.385 , 647.753, 319.127, 582.285], 0, 1, 0, 0, 50, 0, 22  
0, 0)
```

## map\_box and map\_polygon

MapBox and MapPoly objects have the same syntax as the Box and Polygon objects respectively except they include a `fast_preview` parameter before the `page` parameter. The attribute field contains information used to specify the map composition object including the name of the 'legendrules' file and any parameter that the user specifies differently from the default. See *Chapter 19, "Map composition files (.ldd)"* for more information about map objects. For example,

```
map_box("Name =
\"Scale_Bar/Tick\"\\n",0.65749,0.34704,0.8249999,
    0.4011363,0,1,0,-1,-1,-1,0,1).
map_polygon("Name = \"Clip_Mask/Inside\"\\n\\n\"Color\"
= \"253,0, 9\"\\n\"Border Color\" =
\"254,254,254\"\\n\",4,[0.7149999,0.6310227, 0.6874999,
0.6503409,0.8549999,0.6657954,0.8474999,0.5807954],0,
1,0,0,-1,-1,-1,0,1)
```

SPECIFIERS USED IN OBJECT DEFINITIONS			
name	full name	type	comments
angle	angle of text object	real	Angle of text object in degrees (0 - 360) clockwise.
attribute	text attribute or label	alpha numeric characters	Text that will be associated with the object. Must use only alphanumeric characters.
end	line end style	integer	End of line style, where: 0 = plain 1 = arrow - end of line 2 = arrow - beginning of line 3 = arrow - both ends of line
curved	curve	boolean	Set polyline or polygon spline condition on or off. 0 = off (straight lines) 1 = on (curved lines)
font	font	text	The name of the font used in a text object.
fill	fill style	integer	An integer specifying the object fill style. Can be 0 to 19.
just	text justification	integer	Justification of a text object, where: 0 = left 1 = centre 2 = right

<b>SPECIFIERS USED IN OBJECT DEFINITIONS</b>			
<b>name</b>	<b>full name</b>	<b>type</b>	<b>comments</b>
line_of_text	single line of a text object	text including punctuation	The text that shows on the screen or output device. Punctuation is allowed.
ltx	left top x	real	The coordinates of the object, in the units specified in the dataset header file. Raw coordinates are normally in the range zero to a few thousand whereas E/N are in the order of millions eg. 6123123.567.
lty	left top y	real	
rbx	right bottom x	real	
rby	right bottom y	real	
x	x coordinate	real	The coordinates of the bottom left of the first line of text.
y	y coordinate	real	
nlines	number of lines	integer	The number of lines in a text object
npoints	number of points	integer	Number of points in a polyline or polygon
page	page relative or map relative	boolean	0 indicates object coordinates are in image coordinates; 1 indicates object coordinates are relative to the page dimensions.
pen	pen style	integer	An integer specifying the pen pattern. Can be 0 to 23. The styles are those shown in the Annotation line style dialog box.
reserved	reserved	0	Used by ER Mapper. Set to zero.
r, g, b	red, green and blue components of the color	integer -1 to 255	0 to 255 specifies amount of color component. -1,-1,-1 indicates default layer color.
size	text size	real	Size of a text object in points.
spare	spare	0	Not used at this time - set to zero
style	text style	integer	Ignored. Included in font specification.
width	line width	real	The thickness of the line in points.

# Dynamic Links Program Interface

## Introduction

This section provides the user with information regarding the Dynamic Links Program Interface.

## Non-Raster data

Dynamic Links are used to access non-raster data from outside ER Mapper. The data can be of any kind, such as vector data stored in a GIS, tabular data from a spreadsheet or point data from a DBMS.

## ER Mapper PostScript

The language used by ER Mapper for Dynamic Links is PostScript. External data from a source is transformed into ER Mapper PostScript by the appropriate Dynamic Link. This is then further processed and merged by ER Mapper with other Dynamic Link and Raster layers.

## Editable links

ER Mapper also supports Read/Write dynamic links, to enable direct editing of external data in ER Mapper image windows.

## Flexible connections

Dynamic Links are true connections to external systems, querying them for the data required. The complexity of the link and the source and format of the data varies and the differences are accommodated by the flexibility of the Dynamic Link interface. Separate Dynamic Link connections to external data are performed by independent programs which are controlled by a menu file and Dynamic Link manager. To create a new Dynamic Link new programs and a new menu option have to be added.



---

Refer to the ["Dynamic links menu"](#) chapter in this manual.

## Multiple links

Any number of Dynamic Links can be added. Indeed, multiple Dynamic Links to the same system are possible.

## Ready-made links

ER Mapper is supplied with a variety of ready-made links to popular systems and data formats. If there is already a link to the type of data you want to include you can select the appropriate Dynamic Link layer from the menu. If not, a new link must be constructed to transform your data into ER Mapper PostScript format.

## Triggering links

Dynamic Links are triggered each time the user initiates a display (for example by pressing zoom) or printing.

## Examples

The Dynamic Link examples in this chapter have been written in a combination of Shell Script and PostScript and it is assumed that you are familiar with Shell Script programming. Links can however be written in any language as long as they output PostScript to stdout. An example of a Dynamic Link written in "C" is "ermmps\_dxf.c" which creates a Dynamic Link to ".dxf" files. The C source to this program, and the source to other example Dynamic Links, may be found in the "ERMAPPER\examples" directory.

## Georeferenced and Page Relative layers

While ER Mapper can incorporate virtually any type of non-raster data, the type of data displayed falls into one of two distinct categories. These are:

- Georeferenced data
- Page relative data

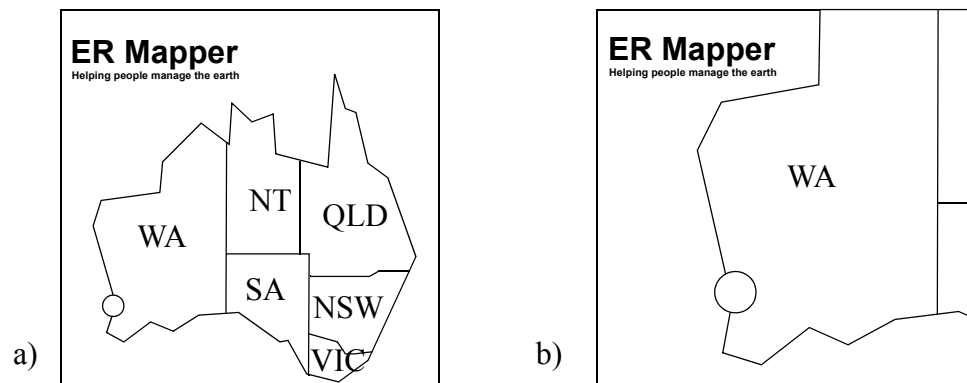
### Georeferenced data

In a Georeferenced layer, data is positioned accurately over a raster image using geographical coordinates. The vector image zooms and moves whenever the raster image is manipulated. You might have vector information such as a road network from a GIS system or pollution density readings at different locations stored in a DBMS table. The Dynamic Links to ARC/INFO are examples of Georeferenced Data layers.

### Page relative layers

In a page relative layer, text or graphics are positioned relative to the image window, not matched to the position of any raster data coordinates. Zooming and changing the window extents have no effect on the positioning of the layer.

All the dynamic links in the **PostScript** submenu in the **Edit / Add Vector Layer** menu are this kind of Dynamic Link.

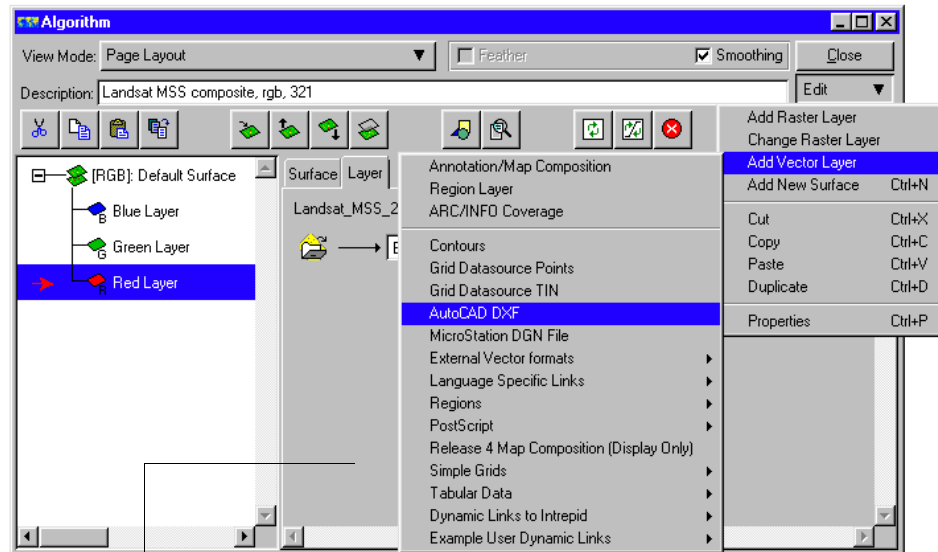


Comparison between georeferenced layers and page relative layers:  
a) The Georeferenced layer showing state boundaries and state names zooms and moves with the raster layer of Australia; b) The Page relative layer with "ER Mapper Helping people manage the earth" always appears in the top left hand corner.

### Dynamic Link layer buttons

Dynamic Link layer options are listed in the lower part of the Edit / **Add Vector Layer** menu in ER Mapper's **Algorithm** dialog box (see below). When an option is chosen from the menu a new layer is added.







Dynamic Link layers



### Dynamic link layer buttons

Dynamic Link layers can have the following buttons associated with them. Parameters in the dynamic links menu file determine which will appear.



*These parameters are referred to briefly below and are described in detail in the section ["Adding the menu option"](#) later in this chapter.*

- For selecting a data source: either the **Dynamic Link chooser** button  for selecting from a list of items or the **Dataset chooser** button  for choosing a data file from a list with file extensions. For example, the "AutoCAD DXF" layer uses the Dataset chooser button to choose a dataset with a ".dxf" extension, while in the "ARC/INFO direct link via ARC/PLOT" layer the Dynamic Link chooser button is used to select from a list of ARC/INFO covers. The "link chooser" parameter specifies which of the two buttons to include or it is possible to omit these buttons altogether if the data source is set specifically in the dynamic link. See ["Chooser program"](#) and ["Link chooser parameter"](#).

- The **Color chooser** button  for selecting the default color of the layer. In some layers this color will be overridden by the linked data.
- The **Edit** button  is only available for read/write dynamic links: Annotation/Map Composition, Tables of data shown as Circles links, Regions links and ARC/INFO Overlay layers are examples. These vector layers can be edited using a variety of drawing tools. The button is specified by the "Edit flag" parameter.

### To use an existing Dynamic Link

Thus, for the ER Mapper user, the procedure for displaying an existing Dynamic Link in an algorithm is to:

- in the **Algorithm** dialog box, choose the appropriate Dynamic Link layer option from the **Edit / Add Vector Layer** menu
- if a **Dataset chooser** button or **Dynamic Link chooser** button is available, use it to select a data source




---

*The use of individual existing Dynamic Links is covered in [Dynamic links \(vector layers\)](#) in the ER Mapper User Guide.*

This chapter details how to construct new Dynamic Links, with their controlling menu option entry and link programs, and using existing links as examples.

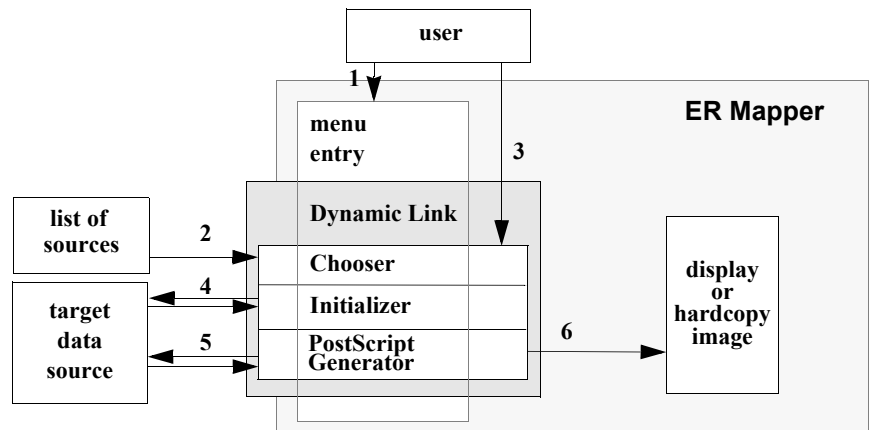
### Inside a Dynamic Link

The previous section looked at Dynamic Links as they appear to the user. In the sections that follow we will look at how a Dynamic Link is constructed.

A Dynamic Link is added as a separate layer. Each link is controlled by an entry in the dynamic link menu file which specifies program calls and other parameters. A Dynamic Link consists of a maximum of 3 procedures. These are:

- Choosing the data source (optional)
- Initializing the link (optional)
- Extracting and converting the data to PostScript.

These procedures are outlined briefly below and detailed in the pages that follow.



### How a Dynamic Link works:

1. The user selects the Dynamic Link menu option, setting up the Dynamic Link;
2. The chooser displays a list of sources;
3. The user selects a source;
4. The Initializer queries the target source about the data extents and map projection;
5. The PostScript Generator asks for and receives the required data, converting it into PostScript;
6. The data, in PostScript format, is merged with other layers and sent to the output device.

### Choosing the data source

In a Dynamic Link, the type of data source is not restricted. Thus ER Mapper must be passed a list of the sources to be specified. For example, all files with a particular extension may be candidates for the link, there may be a set list of possibilities or there may be no choice at all. By the end of this step a particular data source will have been specified. For example, with the ARC/INFO Dynamic links the cover to be extracted is chosen in this step.

### Initializing the link

In this step ER Mapper takes information such as the image extents and current map projection and queries the potential data source for corresponding information. It checks if the data is compatible with other displayed data.

## Extracting the data

The relevant data is converted into PostScript and displayed with any raster and vector layers and other Dynamic Link layers in the current algorithm.

## Creating a Dynamic Link

To create a new Dynamic Link you will need to do the following (in no particular order):

- **add a menu option** to the dynamic links menu file to access and specify the new link from inside ER Mapper
- **write the PostScript generation program** to convert the data to a form compatible with ER Mapper
- if required, **construct a link chooser** so the user can select the data source from a list of options
- where necessary, **write an initialization program** to set up the link.

## ER Mapper PostScript

All Dynamic Links must convert data into PostScript. Thus understanding PostScript interpretation is an important part of creating new Dynamic Links.

In addition, since PostScript is the language on which Dynamic Links are based, it is the logical choice for creating files for display. Many drawing packages can produce their output in PostScript or you can write a file directly in PostScript yourself.

ER Mapper, however, does not interpret PostScript as a fixed page size. Instead it is set up to display images to maximum device resolution and to cope with a changing output canvas size.



---

*These differences are detailed in the chapter on [PostScript](#).*

A number of ready made Dynamic Links are available for accessing PostScript files. There are links set up to handle files output in standard PostScript from other packages and links set up for files written with PostScript interpretation by ER Mapper in mind.

## Looking ahead

The following sections describe:

- *Adding The Menu Option*: to the Dynamic Links menu file
- *Chooser Program*: the alternative ways available for choosing data
- *Link Initialization*: the program interface for the link initializer
- *PostScript Generation*: the program interface for the data extraction and PostScript generation program

## Adding the menu option

- *Example Dynamic Links*: the full code for existing Dynamic Links

Dynamic Link menu options are specified in the ASCII file "dynamiclinks.erm" in the "ERMAPPER\config" directory. The options listed in this file appear in the lower part of the **Edit / Add Vector Layer** menu in the **Algorithm** dialog box. To add your dynamic link option to the menu, you add a single-line menu entry in this file. The entry specifies the data source, link initialization program and PostScript generation program. For example, the entry for the dynamic link to a Microstation DGN file is the following:

```
"MicroStation DGN File" "DGN Link" "ermeps_dgn"  
TRUECOLOR ".dgn" "erminit_dgn"
```

As another example, the entry for an editable link is:

```
"Table Circles with Rotation" "Table of Data with  
Rotation Link" "ermeps_table_circle" MONOCOLOUR  
".tbl" "erminit_any" "EDIT" "erm_link"
```

Briefly the parameters are:

- **Menu option**: The text to be displayed on the menu. In this case:"

Table Circles with Rotation"

- **Description**: A short description of the link shown in the layer in the Algorithm dialog.

"Table of Data with Rotation Link"

- **PostScript generation program**: The program for creating the PostScript for display by ER Mapper.

"ermeps\_table\_circle"

- **PostScript type**: (Optional). Either MONOCOLOUR or TRUECOLOUR.

- **Link chooser**: (Optional). The data source for the dynamic link. ".tbl"

- **Initialization program**: (Optional). A program for setting up the coordinate space and extents for the link.

"erminit\_any"

- **"EDIT"**: (Optional). Indicates a Read/Write (editable) link.

- **8. Edit program**: (Optional, but must be present if 7. is present). The edit program.

"erm\_link"



The 'dynamiclinks.erm' file and these parameters are described in full in the ["Dynamic links menu"](#) chapter.

## Chooser program

### Name and Directory

The link chooser program can be written in any programming language. Those supplied with the earlier Unix versions of ER Mapper were mostly written in shell script. Link initialization programs are typically stored in the \$ERMBIN directory. The program name must not start with "\$" or ".".

### Arguments

No arguments are passed to the chooser program.

### Output

The data chooser program outputs a list of choices to stdout.

The example program below could be used in a dynamic link that requires a distance parameter, such as a grid spacing, to be set. The program passes the list of possible values to the chooser program. One of these is then selected by the user and copied to argument 15.

```
#!/bin/sh
#
# Returns list of possible values
#
cat << EOF
Variable
10 meters
100 meters
500 meters
1,000 meters (1km)
2,500 meters (2.5km)
5,000 meters (5km)
10,000 meters (10km)
25,000 meters (25km)
50,000 meters (50km)
100,000 meters (100km)
1,000,000 meters (1,000km)
EOF
```

A second example extracts the list from a table.

```
#!/bin/sh
#
# 1. Assumes that sqlplus is accessible
# 2. Assumes that headings are turned off in sqlplus
#
# Returns all tables within Oracle accessible to ER Mapper
#
# You can use just about any SQL based logic to select a list of
# tables or views to be displayed under ER Mapper.
#
# This particular example expects to find a table titled
# "ERMAPPER_LAYERS" which will contain one column which will
# be a list of all tables in the format of 3 columns
# (Easting Northing Z-Value) which may be displayed with
# ER Mapper.
#
# You can create any sort of SQL access into Oracle to display
# tabular data; this link is an example of how to do so.

sqlplus << EOF
  SELECT * FROM ERMAPPER_LAYERS
  QUIT
EOF
exit $?
```

### Link initialization

If the sixth parameter in the menu file entry is present, the link initialization program specified is called to check the suitability of the source data. If no program is specified ER Mapper assumes that the source data is always compatible.

The link initialization program runs at some time after the chooser program and before the PostScript generation program. It may be called by ER Mapper any time the data compatibility may have changed, for example, when a layer in a different coordinate system has been switched on.

### Name and Directory

The link initialization program can be written in any programming language. It is typically stored in the "ERMBIN" directory. The program name must not start with "\$" or ".".

## Arguments

ER Mapper always passes 15 arguments to the link initialization program which may use them or ignore them as required. These arguments are also passed to the PostScript generation program (see next section). This means that the same program can be used for both link initialization and PostScript generation. When the program is being called for initialization the first field is set to "mapinfo".

The fifteen arguments are:

Number	Name	Example	Comment
1	COMMAND	"mapinfo"	"mapinfo" for link initialization "postscript" for PostScript generation
2	DATUM	"AGD66"	Datum, from the coordinate database
3	PROJECTION	"TMAMG54"	Projection, from the coordinate database
4	COORDTYPE	"EN"	"EN", "LL" or "RAW"
5	UNITS	"METERS"	The preferred units
6	ROTATION	0	Rotation from North in radians
7	TLX	100	Top Left X value in x/easting/latitude
8	TLY	200	Top Left Y value in y/northing/longitude
9	BRX	300	Bottom Right X in x/easting/latitude
10	BRY	400	Bottom Right Y in y/northing/longitude
11	CANVASWIDTH	500	Current canvas width in pixels
12	CANVASHEIGHT	600	Current canvas height in pixels
13	DPIX	83.1272727	Device X dots per inch
14	DPIY	83.1272727	Device Y dots per inch
15	FILESPEC	"Roads"	File name or user choice or fixed argument from link chooser step

## Output

The link initialization program outputs 11 fields to STDOUT. The first ten fields are the same as those passed to the program, while the eleventh is optional and should only be specified if the link is capable of handling more than one coordinate system.

Fields must be separated by a space (or tab) and any fields containing imbedded blanks must be enclosed in quotes ("). Quotes are not permitted within any argument.



The arguments returned from the link initialization program are:

Number	Name	Example	Comment
1	COMMAND	"mapinfo"	"mapinfo" for link initialization "postscript" for PostScript generation
2	DATUM	"AGD66"	Datum, from the coordinate database
3	PROJECTION	"TMAMG54"	Projection, from the coordinate database
4	COORDTYPE	"EN"	"EN", "LL" or "RAW"
5	UNITS	"METERS"	The preferred units
6	ROTATION	0	Rotation from North in radians
7	TLX	100	Top Left X value in x/easting/latitude
8	TLY	200	Top Left Y value in y/northing/longitude
9	BRX	300	Bottom Right X in x/easting/latitude
10	BRY	400	Bottom Right Y in y/northing/longitude
11	MULTI-C SPACES	MULTI	Can handle multiple coordinate spaces flag

## Errors

Errors within the link initialization program can be reported by printing an error message to STDERR and exiting the program with a non-zero exit code. ER Mapper will check the exit code of the initialization program and if it is non-zero it will look for a message in STDERR.

## Example initialization program

An example link initialization program "erminit\_any" from the "\$ERMBIN" directory (ERMAPPER\bin\win32). This program simply echoes the link arguments back to ER Mapper.

```
void main(int argc, char **argv)
{
    int i;
    /* Make sure we have the correct count for dynamic
    link arguments */
    if(argc < 16){
        fprintf(stderr, "Expected 15 arguments got
%d\n", argc - 1);
        exit(1);
    }

    /* output the first 5 arguments with quotes */
    for(i = 1; i < 6; i++)
        fprintf(stdout, "\"%s\" ", argv[i]);

    /* output the last 7 without quotes */
```

```

        fprintf(stdout, "%s /*rotation*/
                %s      /*top x*/
                %s      /*top y*/
                %s      /*bottom x*/
                %s      /*bottom y*/
                %s      /*dpi x*/
                %s      /*dpi y*/

        "MULTI", argv[6], argv[7], argv[8], argv[9], argv[10],
                argv[13], argv[14]);

        fflush(stdout);

        exit(0);

    }

```

## PostScript generation

ER Mapper calls the PostScript conversion program to create PostScript. The program name is specified in the third parameter in the entry in the dynamic link menu file. This is the main part of the link. The program extracts data from the source specified in the link chooser stage and converts it into PostScript so that it can be displayed by ER Mapper. For example, the 'ermeps\_dxf' program reads an AutoCAD DXF file and converts it into PostScript which may be integrated with other data by ER Mapper.

This section describes how to create this kind of PostScript generation program. It is assumed you have some knowledge of the PostScript language.




---

*More detailed information about writing PostScript for ER Mapper can be found in the ["Postscript"](#) chapter of this manual.*

### Name and Directory

By convention, the names of PostScript conversion programs begin with "ermeps\_" and the programs are stored in the "\$ERMBIN" directory. The program can be written in any programming language (shell script and C are common choices).

### Arguments

ER Mapper passes 15 arguments to the PostScript generation program, which can use or ignore them as required. These are the same arguments that are passed to the link initialization program, with the first argument is set to "postscript". See the section ["Link initialization"](#) earlier in this chapter for a full description of the parameters.

Of particular interest are the following parameters:

- CANVASW (argument 11) - the width in pixels for the canvas

- CANVASH (argument 12) - the height in pixels for the canvas
- DPIX (argument 13) - the DPI in X for the output device
- DPIY (argument 14) - the DPI in Y for the output device

When creating your output PostScript, you can assume that you are rendering into an output canvas which is CANVASW pixels across and CANVASH pixels down, with an origin of (0,0) at the bottom-left hand corner of the canvas.

Thus, the PostScript code:

```
%!  
0 0 moveto  
WidthI HeightI lineto  
stroke
```

will draw a line from the bottom-left hand corner of the image to the top-right hand corner of the image. You should always start your PostScript output code with the characters "%!".

Notice the use of two variables "WidthI" and "HeightI". These are defined within the PostScript dictionary by ER Mapper, allowing an application to know the size in device pixels for the output image.



---

*These and other variables are described in the ["Postscript"](#) chapter of this manual.*

## **Output**

The PostScript generation program returns PostScript to standard output (STDOUT).

## **Errors**

Errors within the PostScript generation program can be reported by printing an error message to STDERR and exiting the program with a non-zero exit code. ER Mapper will check the exit code and if it is non-zero it will look for a message in STDERR.

A simple example of a PostScript generation program, "ermips\_date" is given below. It is written in C and outputs PostScript to STDOUT.

```
** PURPOSE:To Produce the date and time in the bottom left side of the image
#ifndef lint
static char *sccsid=@"(#)%M%:%I%%D%Copyright Earth Resource Mapping
1989-95";
#endif

#include "ERS.h"
#include <time.h>

void main(int argc,char **argv[])
{
    char *p_ascitime;
    struct tm *p_tm;
    time_t tm;

    time(&tm);

    p_tm = localtime(&tm);

    p_ascitime = asctime(p_tm);

    fprintf(stdout,"%%!PS-Adobe-1.0\n"
        "%%\n"
        "/FS HeightI 20 div def"
        " %% font size = 1/20th of image height\n"
        "0 setgray          %% set gray level (== black)\n"
        "%% <Date>\n"
        "/Helvetica findfont FS 0.4 mul scalefont setfont\n"
        "FS FS FS .6 mul sub moveto\n"
        "%s show\n"
        "showpage\n",p_ascitime);

    exit(0);
}
```

## Printing dynamic links

Dynamic Links are fully supported in hardcopy generation. The ER Mapper hardcopy engine may have to strip print "tiles" or "strips" of the image if the user requests an image that is larger than the hardcopy device. The entire strip printing issue is hidden from Dynamic Links. The following points may be of interest:

- Your Dynamic Link will be called once only to create the PostScript for the entire image in the layer, regardless of how many hardcopy strips are required to create that image.

- Your PostScript code may be rendered into multiple canvases, one for each strip (and indeed one for each sub-strip if strips are larger than available memory for the PostScript engine). However, at all times your PostScript code “sees” the entire image.

## MONOCOLOUR vs TRUECOLOUR links

PostScript can be either MONOCOLOUR or TRUECOLOUR. The type of PostScript is specified in the menu entry for each Dynamic Link. In most cases you will only want to use MONOCOLOUR. This means that the entire layer will be displayed in a single color which is chosen by the user using the Color button.

In TRUECOLOUR PostScript the color is specified in the PostScript code itself and any number of colors can be used in the one layer. However, this kind of PostScript takes longer to process, so use MONOCOLOUR wherever possible.

If TRUECOLOUR PostScript is linked using a MONOCOLOUR link, it is displayed as if it were in monochrome in the color specified by the Color button.

## Batch Script Chooser

With this option you can specify a batch script to run to determine the source of the data. The syntax is:

```
"$$SCRIPT=batchfile.erb"
```

You can also use the \$DEFAULT keyword which is replaced with the last value returned by the batch script. The syntax is:

```
"$$SCRIPT=batchfile.erb $DEFAULT"
```




---

Refer to the [“Batch scripting and wizards”](#) chapter for more information.

## Debugging dynamic lines

To assist you with creating and debugging new Dynamic Links there are a number of links which echo back the values of the link parameters and display them on the screen. These can be found on the Example User Dynamic Links submenu. They are:

- Show arguments for No Parameter
- Show arguments for Dataset Chooser
- Show arguments for Fixed Parameter
- Show arguments for \$\$ALG Parameter
- Show arguments for Link Chooser
- Show arguments for External Link Chooser
- Show arguments for Script Link Chooser



The links correspond to the categories of syntax for the link chooser parameter in the dynamic links menu file (see the ["Dynamic links menu"](#) chapter).

The Dynamic Link menu file entries for these links are shown below.

```
"Example User Dynamic Link" "Example User Dynamic Link" "ermeps_example"
MONOCOLOUR "$ER Mapper"
"Show arguments for No Parameter" "No Parameter Link" "ermeps_info" MONOCOLOUR
"Show arguments for Dataset Chooser" "Dataset Chooser Link" "ermeps_info"
MONOCOLOUR ".ers" "erminit_ers"
"Show arguments for Fixed Parameter" "Fixed Parameter Link" "ermeps_info"
MONOCOLOUR "$variable"
"Show arguments for $$ALG Parameter" "$$ALG Parameter Link" "ermeps_info"
MONOCOLOUR "$$ALG"
"Show arguments for Link Chooser" "Link Chooser Link" "ermeps_info" MONOCOLOUR
"erm_choose_en_grid"
"Show arguments for External Link Chooser" "External Link Chooser Link"
"ermeps_info" MONOCOLOUR "$$CHOOSER=erm_xgettext -s 30 -p \"Enter some text\" -
name \"External Link Chooser\""
"Show arguments for Script Link Chooser" "Script Link Chooser Link" "ermeps_info"
MONOCOLOUR "$$SCRIPT=Dlink_Chooser.erb $DEFAULT"
```

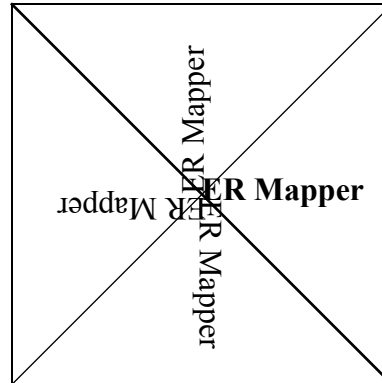
To use these links to develop your own dynamic link, substitute your link chooser and initialization program names for the fourth and fifth arguments of the appropriate link entry. For example, to see the link parameters passed to the "MicroStation DGN File", substitute the link chooser ".dgn" and the initialization program called "erminit\_dgn" for the fourth and fifth arguments in the "Show arguments for Link Chooser" Dynamic Link menu entry. Change the PostScript type from MONOCOLOUR to TRUECOLOR. If a Chooser button is available as part of your link you will have to select a data source to display the arguments.

<b>"MicroStation DGN File" menu entry in ERMAPPER\config\dynamiclinks.erm</b>					
1. Menu option	2. Description	3. PostScript generation program call	4. PostScript type	5. Link chooser (optional)	6. Initialization program call (optional)
"MicroStation DGN File"	"DGN link"	"ermeps_dgn"	TRUECOLOR	".dgn"	"erminit_dgn"
<b>"Show arguments for Link Chooser" menu entry in ERMAPPER\config\dynamiclinks.erm</b>					
1. Menu option	2. Description	3. PostScript generation program call	4. PostScript type	5. Link chooser (optional)	6. Initialization program call (optional)
"Show arguments for Link Chooser"	"debug dynamic link"	"ermeps_info"	TRUECOLOR	".dgn"	"erminit_dgn"

### Example 1 - Example User Dynamic Link

This is an example of a link which passes a string from the dynamic link menu file to the PostScript generation program. The string specified in the fifth parameter in the menu entry is inserted into a program written in C and ER Mapper PostScript. This is a page referenced layer that draws on the screen without taking the coordinate system or data extents into account. There is no Chooser button available for the layer because the layer is generated by the PostScript generation program using the text "ER Mapper" passed by parameter five.

The link draws a box on the screen with the word "ER Mapper" rotated through the center as follows (actually rotated twelve times).



This layer may be displayed by selecting the **Example User Dynamic Links / Example User Dynamic Link** option from the **Edit / Add Vector Layer** menu in the **Algorithm** window. This option calls the Dynamic Link filter program 'ermeps\_example' to create the PostScript code for this link. There is an example algorithm

**Functions\_And\_Features\Dynamic\_Links\User\_Example\_Dynamic\_Link** which uses this example Dynamic Link.

The complete code for this example link is given below. It has been implemented in C to demonstrate just how easy it is to create a Dynamic Link to ER Mapper. More sophisticated examples may be found in the **ERMAPPER\examples** directory; in particular the **ermeps\_dxf.c** program which creates a Dynamic Link to DXF vector files.

The **ermeps\_example** Dynamic Link will print any word as the text. Change the fifth parameter in the **Example User Dynamic Link** menu entry in the **dynamiclink.erm** configuration file.

It should be noted that this example Dynamic Link always draws a box the size of the image whereas most Dynamic Links extract and display a particular region of interest which changes as the ER Mapper user zooms in and out.

## Dynamic link menu entry in 'dynamiclinks.erm'

Dynamic link menu entry ERMAPPER\config\dynamiclinks.erm				
1. Menu option	2. Description	3. PostScript generation program call	4. PostScript type	5. Link chooser (optional)
"Example User Dynamic Link"	""	"ermps_example"	MONOCO LOUR	"\$ER Mapper"

## PostScript generation program 'ermps\_example'

```

/*****
**
** PURPOSE:
** An example dynamic link. This one draws a box the
** size of the
** image with a cross through it, and the rotates the
** word
** "FileSpec" (arg 15) through the image. More advanced
** examples
** such as "ermps_dxf.c" may be found in
** $ERMAPPER/examples
**
** Arguments
** 1 COMMAND      # should be "postscript"
** 2 DATUM        # geodetic datum name
** 3 PROJECTION   # projection name
** 4 COORDTYPE    # type of coordinates (EN, LL or RAW)
** 5 UNITS        # Units (eg. METERS)
** 6 ROTATION     # rotation
** 7 TLX          # top left x coordinate
** 8 TLY          # top left y coordinate
** 9 BRX          # bottom right x coordinate
** 10 BRY         # bottom right y coordinate
** 11 CANVASWIDTH # window width (0 on init)
** 12 CANVASHEIGHT# window height (0 on init)
** 13 DPIX=$4     # x dots per inch (0 on init)
** 14 DPIY=$5     # y dots per inch (0 on init)
** 15 FILESPEC   # file spec or choice string
**
** EDITS:
**
*****/

#ifdef lint
static char *sccsid="@(#)M%: %I%D%Copyright Earth
Resource Mapping 1989-96";
#endif

#include "ERS.h"

```



```

#include "ps_util.h"

int main(int argc, char **argv)
{
    erm_dlinkargs dlink;

    if(!erm_initdlinkargs(argc,argv,&dlink))
        exit(1);

    if(!dlink.filespec ||
        (dlink.filespec && !strlen(dlink.filespec)))
    {
        fprintf(stderr, "%s: Filespec argument is null",
            argv[0]);
    }

    /* Now generate the postscript code */

    fprintf(stdout, "%!\n"
        "% A simple example ER Mapper Dynamic Link\n"
        "%This draws a box the exact size of the
resultant image\n"
        "%that ER Mapper is to generate, with a cross
through it.\n"
        "%It then draws some fancy rotated text\n"
        "\n"
        "% value was in 72 point\n"
        "/from72pt { %lf 72 div mul } def\n"
        "\n"
        "% First, draw a cross through the image
window\n"
        "% make sure is always 10 point (72 dpi) wide\n"
        "% regardless of device actual dpi\n"
        "2 from72pt setlinewidth\n"
        "0 0 moveto\n"
        "0 0 moveto\n"
        "WidthI 0 lineto\n"
        "WidthI HeightI lineto\n"
        "0 HeightI lineto\n"
        "0 0 lineto\n"
        "stroke\n"
        "0 0 moveto\n"
        "WidthI HeightI lineto\n"
        "stroke\n"
        "WidthI 0 moveto\n"
        "0 HeightI lineto\n"
        "stroke\n"
        "\n"

```

```

        %% now draw the variable (arg 15) as rotated
text\n"
    %% Text will always be this point regardless of
image"
    %% size\n"
    /Helvetica-Bold findfont 32 from72pt scalefont
setfont \n"
    "\n"
    /oshow %% stack: grey (string)\n"
    "{\n"
        "true charpath \n"
        "gsave setgray fill grestore\n"
        "stroke\n"
    } def\n"
    "\n"
    /circleofERM\n"
    "{\n"
        "30 30 330\n"
        "{\n"
            "gsave\n"
            "dup 345 divexch %% for the setgray\n"
            "rotate 0 0 moveto\n"
            "(%s) oshow\n"
            "grestore\n"
        } for\n"
    } def\n"
    "\n"
    %% begin program\n"
    WidthI 2 div HeightI 2 div translate\n"
    "\n"
    ".5 from72pt setlinewidth\n"
    "circleofERM\n"
    "0 0 moveto\n"
    "(%s) show\n"
    "\n"
    "showpage\n",
    dlink.dpix,
    dlink.filespec,
    dlink.filespec);
exit(0);
}

```



*In a monocolour overlay the color is set by the color button. So in the code above, 'black' means full color and 'white' is no color.*

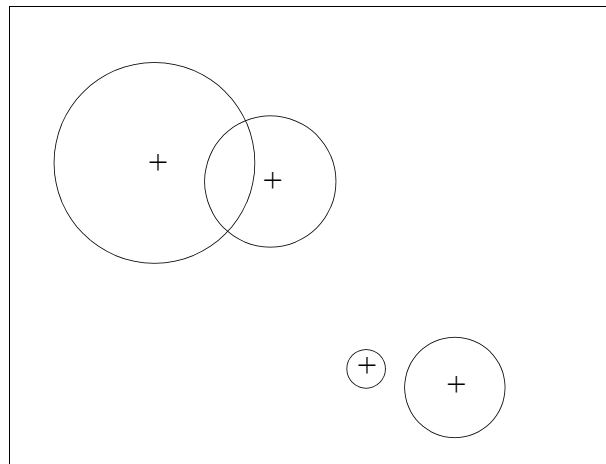
### **Example 2 - Table of data shown as circles**

This link reads a file containing a table with three columns which are an Easting, a Northing and a Value. For each row of data, the link draws a circle of diameter Value at the Easting-Northing position.

## Output

The figure below shows the output for a sample table of data. In reality the table would not have a heading row.

<b>eastings</b>	<b>northings</b>	<b>value</b>
<b>461200</b>	<b>620000</b>	<b>0.85</b>
<b>462900</b>	<b>619800</b>	<b>0.65</b>
<b>463200</b>	<b>618000</b>	<b>0.25</b>
<b>464100</b>	<b>617800</b>	<b>1.3</b>



The standard ER Mapper dataset chooser is called because of the **".ext"** syntax of the fifth parameter. All **".tbl"** files stored in the **\$ERMDS DIR** path are listed by the Chooser.

<b>Dynamic link menu entry ERMAPPER\config\dynamiclinks.erm</b>					
1. Menu option	3. PostScript generation program call	4. PostScript type	5. Link chooser (optional)	6. Initialization program call (optional)	7. Edit flag (optional)
"Table of data shown as Circles"	"ermeps_table_circle"	MONOC OLOUR	".tbl"	"erminit_en0"	

## Initialization

The Link Initialization program tells ER Mapper that the link is only compatible with an EN coordinate system by setting the projection and unit arguments to appropriate values. It does not actually look at the table file but assumes that the values are Eastings-Northings. The projection argument cannot simply be echoed back because, if this is the first layer, the projection will be set to default RAW projection. The other arguments are simply echoed back so the entire table is used regardless of the data extents. Note that all fifteen arguments are read in, even though they are not all used, and eleven arguments are output to STDOUT as required. The program is written in shell script.

Link initialization program ERMAPPER\bin\win32\erminit\_en0

```
/*
*****
*****
** PURPOSE:This program accepts the arguments passed
from ER Mapper, checks
** that the correct number of arguments have been passed
(i.e. 16), then outputs
** the arguments in the correct format ready for
processing by the PostScript
** generation program.

*****
*****/

int main(argc, argv)
int argc;
char **argv;
/*
** This program accepts the arguments passed from ER
Mapper to the link, checks
** that there are the correct number of arguments (i.e.
16), then outputs the
** passed arguments (if given the number of arguments)
to stdout ready for use
** by the PostScript generation program.
*/
{

    get_link_initdata(argc, argv);

    if (strcasecmp (datum,"RAW") == 0) {
        safe_free(datum);
        datum = strsave("AGD66");
        safe_free(projection);
        projection = strsave("TMAMG54");
    }
}
```

```

        safe_free(coordtype);
        coordtype = strsave("EN");
#ifdef ERMINIT_NOTRAW
#else
        rotation = 0; /* Sets rotation to 0 for RAW case */
#endif
        safe_free(units);
        units = strsave("METERS");/*
                                ** Always assume units for coordinate
                                ** space are meters.
                                */

        put_link_initdata();
        exit(0);
        /* NOT REACHED */
    }

#define CORRECT_NR_ARGS 16

static void get_link_initdata(argc, argv)
int argc;
char **argv;
/*
** This function checks that the correct number of
** arguments have been
** passed to the initialisation program, strips quotes
** from any argu-
** ment that has them, then saves each argument into a
** variable with
** a meaningful name.
*/
{
    INT32 i;
    char *p_c;
    STRING base_progname;

    progname = argv[0];

    /* Check link was passed correct number of arguments
    */
    if (argc != CORRECT_NR_ARGS) {
        error("Wrong number of arguments, expected %d,
received %d.",
            CORRECT_NR_ARGS, argc);
        /* NOT REACHED */
    }

    /* Strip quotes off any arg that has them */
    for (i = 0; i < argc; i++) {

```

```

        p_c = argv[i];
        if (*p_c == '"') {
            argv[i] = ++p_c;
            while (*p_c != '\\0' && *p_c != '"') {
                p_c++;
            }
            *p_c = '\\0';
        }
    }

    /* Save passed arguments into variables with
meaningful names */
    command = strsave(argv[1]);
    datum = strsave(argv[2]);
    projection = strsave(argv[3]);
    coordtype = strsave(argv[4]);
    units = strsave(argv[5]);
    rotation = atof(argv[6]);
    tlx = atof(argv[7]);
    tly = atof(argv[8]);
    brx = atof(argv[9]);
    bry = atof(argv[10]);
    canvas_width = atof(argv[11]);
    canvas_height = atof(argv[12]);
    dpi_x = atof(argv[13]);
    dpi_y = atof(argv[14]);
    filespec = strsave(argv[15]);
}

static void put_link_initdata(void)
/*
** This function outputs the (validated) arguments
passed to the link
** to standard output ready for processing by the
PostScript generation
** program.
*/
{
    output_string(command);
    output_string(datum);
    output_string(projection);
    output_string(coordtype);
    output_string(units);
    output_double(rotation);
    output_double(tlx);
    output_double(tly);
    output_double(brx);
    output_double(bry);
    output_double(dpi_x);
    output_double(dpi_y);
}

```

```

        output_string("MULTI");

#ifdef DEBUG
        fprintf(tfp2, "\n");
        fclose(tfp2);
#endif
    }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
*/

static void output_string(s)
STRING s;
/*
** This function outputs a string argument in the
correct format for use
** by the PostScript generation program.
*/
{
    fprintf(stdout, "\"%s\" ", s);
#ifdef DEBUG
    fprintf(tfp2, "<%s>", s);
#endif
}

static void output_double(d)
double d;
/*
** This function outputs a double-precision floating
point argument in the
** correct format for use by the PostScript generation
program.
*/
{
    fprintf(stdout, "%.17G ", d);
#ifdef DEBUG
    fprintf(tfp2, "%.17G ", d);
#endif
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
*/

/* VARARGS */
static void error(char *format, ...)
/*
** Simple function to output error messages encountered
when initialising
** link.
*/

```

```

{
    va_list args;

    fprintf(stderr, "%s: ", progname);
    va_start(args, format);
    vfprintf(stderr, format, args);
    va_end(args);

    exit(1);
}

```

## Displaying the data

The "Example" below, which illustrates scaling of EN coordinates to pixel values, is based on this PostScript generation program. Where that example only draws one circle, at a specified point the program below reads a table of data.

PostScript generation program  
ERMAPPER\bin\win32\ermpps\_table\_circle

```

/*****
*****
** PURPOSE:This is a sample dynamic link PostScript
generation program for
** drawing circles defined by data read from a simple
ASCII data file.
**
** The program reads an ASCII file containing data of
the form:
**
**          x, y, r, <text>
**
** and generates PostScript output for displaying
circles of radius r at the
** location (x,y) in an ER Mapper image window.
**
** NOTES:
**      - Coordinates are assumed to be in meters.
**
*****
*****/

#ifdef lint
static char *scsident="@(#)M%:I%D%Copyright Earth
Resource Mapping 1989-98";
#endif

#include "ERS.h"
#include "ps_util.h"
#include <stdio.h>

```



```

void main(int argc, char **argv)
/*
** Main program for dynamic link PostScript generation
program.
*/
{
    erm_dlinkargs dlink;
    FILE *fd;
    char line_buf[BUFSIZ];
    int circles = TRUE;

#if !defined(win32)
    char str[255];
#endif

    /* Used to determine whether to generate filled
circles or not */
    if(strcasecmp(argv[0], "ermmps_table_outline") == 0)
        circles = FALSE;
    else
        circles = TRUE;

    /*
    ** Save the arguments passed by ER Mapper into the
"dlink"
    ** structure ready for processing.
    */
    if(!erm_initdlinkargs(argc, argv, &dlink))
        exit(0);

    if(dlink.filespec == NULL || *dlink.filespec ==
'\0'){
        fprintf(stdout, "Filespec is null \n");
        exit(0);
    }

    /* Open the file passed from the initialisation
program as last argument */
    if((fd = fopen(dlink.filespec, "r")) == NULL){
        fprintf(stdout, "Could not open table:
%s\n", dlink.filespec);
#if !defined win32
        fprintf(stdout, "System Error : %d
%s\n", erm_errno(),
#ifdef sun4
sys_errlist(erm_errno()));
#else

```

```

strerror(erm_errno()));
#endif
#endif
    exit(0);
}

/*
** Write out the required PostScript definitions
*/

/* Write out PostScript header */
fprintf(stdout,
    "%!PS-Adobe-1.0\n"
    "% Table Circle dynamic link for table file
$FILESPEC\n"
    "% Args: $*\n"
    "%\n");

fflush(stdout);

if (!strcmp(dlink.projection, "GEODETTIC"))
{
    fprintf(stdout,
        "/x_map {\n"
        "dup %lf gt\n"
        "  { 360.0 sub }\n"
        "  { dup %lf lt\n"
        "    { 360.0 add } if\n"
        "  }\n"
        "ifelse\n"
        "%lf sub\n"
        "} def\n",
        dlink.brx,
        dlink.tlx,
        dlink.tlx
    );
}
else
    fprintf(stdout, "/x_map { %lf sub }
def\n", dlink.tlx);

# fprintf(stdout,
    "/x_scale { %d %lf %lf sub div } def\n"
    "/y_scale { %d %lf %lf sub div } def\n",
    dlink.canvaswidth,
    dlink.brx,
    dlink.tlx,

```

```

        dlink.canvasheight,
        dlink.tly,
        dlink.bry
    );

    /* Write out PostScript definition for circle
    outline. */
    fprintf(stdout,
        "/do_outline {%% e n value\n"
        "gsave\n"
        "3 1 roll%% get Northing (Y)\n"
        "%lf sub \n"
        "y_scale mul\n"
        "exch %% get Easting (X)\n"
        "x_map\n"
        "x_scale mul\n"
        "exch\n"
        "translate\n"
        "gsave\n"
        "1 from72pt setlinewidth\n"
        "-5 from72pt 0 moveto\n"
        " 5 from72pt 0 lineto\n"
        " 0 from72pt -5 from72pt moveto\n"
        " 0 5 from72pt lineto\n"
        "stroke\n"
        "grestore\n"
        "newpath\n"
        "%lf mul 2 div\n"
        "0 exch 0 exch\n"
        "2 from72pt setlinewidth\n"
        "0 360 arc stroke\n"
        "grestore\n"
        "} def\n",
        dlink.bry,
        dlink.dpix
    );

    /* Write out PostScript definition for drawing
    filled circle. */
    fprintf(stdout,
        "/do_circle {%% e n value\n"
        "gsave\n"
        "3 1 roll%% get Northing (Y)\n"
        "%lf sub \n"
        "y_scale mul\n"
        "exch %% get Easting (X)\n"
        "x_map\n"
        "x_scale mul\n"
        "exch \n"
        "translate\n"

```

```

        "newpath\n"
        "%lf mul 2 div\n"
        "0 exch /cradius exch def 0 exch cradius \n"
        "0 360 arc \n"
        "%0.5 setgray fill\n"
        "0.0 setgray eoclip\n"
        "0.5 setlinewidth\n"
        "/l_length cradius cradius mul 2 mul sqrt cvi
2 mul def\n"
        "cradius neg cradius neg translate\n"
        "/X 0 def\n"
        "/Y cradius 2 mul def\n"
        "cradius 2 mul cvi {\n"
            "X Y moveto\n"
            "l_length l_length rlineto\n"
            "stroke\n"
            "/Y Y 2 sub def\n"
        "} repeat\n"
        "grestore\n"
    "} def\n"
    "%%\n",
    dlink.bry,
    dlink.dpix
    );
/*
** Now read the data from input and generate the
output PostScript file
** for specifying which objects to draw.
*/
while(!feof(fd) && fgets(line_buf, BUFSIZ, fd) !=
NULL){
    if(line_buf[0] && line_buf[0] != '#' &&
        line_buf[0] != '\n')
    {
        double x, y, size;
        sscanf(line_buf, "%lf%lf%lf", &x, &y, &size);

        /*
filled    ** If code called as ermps_table_circle then
        ** circles are drawn.
        */
        if (circles)
            fprintf(stdout, "%lf %lf %lf do_circle\n",
                x, y, size);
        else
            fprintf(stdout, "%lf %lf %lf do_outline\n",
                x, y, size);
    }
}

```

```

    }

    fclose(fd);

    fprintf(stdout, "%s", "%% end of postscript\n");

    fflush(stdout);

    exit(0);
}

```

### Example 3 - Dynamic Link to ARC/INFO

This link accesses ARC/INFO and extracts a cover. It passes ARC/INFO the page size and image extents and ARC/INFO generates PostScript for the appropriate area, scaled to the requested page size. Linking to a system that generates PostScript output is ideal because very little work has to be done by the link itself, only accessing the system and converting the PostScript to ER Mapper format. This Dynamic Link simply extracts a plot file but could easily be edited to use more complex ARC/INFO and ARC/PLOT processing.

Dynamic link menu entry \$ERMAPPER/config/dynamiclinks.erm					
1. Menu option	3. PostScript generation program call	4. PostScript type	5. Link chooser (optional)	6. Initialization program call (optional)	7. Edit flag (optional)
"Dynamic Link to ARC/INFO"	"ermpps_arc"	MONOCO LOUR	"erm_arc_layer"	"erminit_en0"	

### Choosing a Layer

The link chooser program changes to the directory holding ARC/INFO covers, finds the list of covers and sends them to STDOUT to be picked up and displayed by the standard link chooser.

Link chooser program \$ERMBIN/erm\_arc\_layer

```

#!/bin/sh
#
# Returns all ARC/INFO covers in the workspace
#
# This must return a list of the layers available to the
user.
#
# In ARC/INFO terms, a layer is a "cover" within a
workspace, so we
# simply return all the covers within the workspace that
is accessible
# by ER Mapper (via the ARC_WORKSPACE value in the
site.erm configuration file).
#
# If the value ARC_REMOTE is defined in site.erm, then
it is assumed that

```

```

# ARC/INFO is running on a remote machine, and
ARC_REMOTE is used to log onto
# that machine (typically a rsh command) and
ARC_REMOTE_ARC will run the
# remote arc info.
#
# In both local and remote cases it assumed that
ARC/INFO is running on a
# Unix system. This shell script needs to be changed if
it is not.
# gets the workspace name from the site.erm file
ARC_WS=`$ERMBIN/erm_menu/erm_qsite ARC_WORKSPACE`
if [ $? -ne 0 ]
then
    echo "The ARC/INFO Workspace to use must be defined
in site.erm" 1>&2
    # generates error message if workspace name is not
found
    exit 1
fi
# changes workspace name to lowercase
ARC_WS=`echo $ARC_WS | tr "[A-Z]" "[a-z]"`
# gets the remote host name from the site.erm file
ARC_REMOTE=`$ERMBIN/erm_menu/erm_qsite ARC_REMOTE`
if [ $? -eq 0 ]
then
    # runs this if a remote host is found
    # remote machine is assumed to be a Unix system
    # we don't use the LISTCOVERAGES because of the
header stuff it hands back,
    # so find is used instead.
###    $ARC_REMOTE $ARC_ARC LISTCOVERAGES $ARC_WS
NOSTATUS    ### DO NOT USE
# accesses the remote host, changes directory, finds the
list of
# covers and sends it to stdout in uppercase
    rsh $ARC_REMOTE "cd $ARC_WS; find . -name '*' -prune
-print" \ |
    awk -F\ / '{ print $2 }' | \
    tr "[a-z]" "[A-Z]"
else
# else if remote host name is not found the local
version of the
# previous command is executed
    cd $ARC_WS; find . -name '*' -prune -print | \
    awk -F\ / '{ print $2 }' | \
    tr "[a-z]" "[A-Z]"
###    $ARC_ARC LISTCOVERAGES $ARC_WS NOSTATUS
### DO NOT USE

```

## Initializing

The link initialization program is the same as in the previous example and will not be repeated here. It could be improved to query ARC/INFO for the data extents.

## Generating output

The PostScript generation program asks for PostScript output from ARC/INFO and displays the results.

PostScript generation program \$ERMBIN/ermps\_arc

```
#!/bin/sh                                # start of shell script
#
# Returns a ARC/INFO cover in PostScript
#
# In ARC/INFO terms, a layer is a "cover" within a
workspace, so we
# run arcplot to generate a PostScript plot for the
region and cover selected.
#
# The following must be defined in site.erm:
#
# ARC_WORKSPACE - the ARC/INFO workspace to use
# ARC_ARC       - the ARC/INFO arc command
# ARC_REMOTE    - (optional) the remote machine ARC/INFO
is running on
#
# In both local and remote cases it assumed that
ARC/INFO is running on a
# Unix system. This shell script needs to be changed if
it is not.
#
# Arguments (see ER Mapper manuals for full details):
# 1    2    3    4    5    6    7    8    9    10
# 11   12   13   14   15
# cmd datum proj'n coord units rotation tlx tly brx bry
canvasw canvash dpiX dpiY file
#
# checks for 15 arguments
if [ $# -ne 15 ]
then
    echo "$0: Expected 15 Dynamic Link arguments." 1>&2
    exit 1
fi
# defines shell procedure output_header which cats
following postscript to stdout
output_header() {
    cat <<-EOF
    %!PS-Adobe-1.0                % start of postscript
    % PostScript dynamic link for file $FNAME
    % define procedure to convert inches to points
```

```

% hide ER Mapper variable device height...
/inch {72 mul} def

% scale to fit - scale all dimensions by
pixels/points factor to convert standard
% postscript to ER Mapper postscript
WidthI $PAGE_WIDTH inch div HeightI $PAGE_HEIGHT
inch div scale
%
%
-20 -15 translate % for ARC/INFO 5 only - removes
offset
EOF % end of postscript
}
# Grab the arguments and set them up in meaningful
variable names.
# reads arguments into variables
COMMAND=$1 # should be "postscript"
DATUM=$2 # geodetic datum name
PROJECTION=$3 # projection name
COORDTYPE=$4 # type of coordinates
(EN, LL, or RAW)
UNITS=$5 # Units (eg: METERS)
ROTATION=$6 # rotation
TLX=$7 # top left x coordinate
TLY=$8 # top left y coordinate
BRX=$9 # bottom right x coordinate
shift 9
BRY=$1 # bottom right y coordinate
CANVASWIDTH=$2 # window width (0 on init)
CANVASHEIGHT=$3 # window height (0 on init)
DPIX=$4 # x dots per inch (0 on init)
DPIY=$5 # y dots per inch (0 on init)
FILESPEC=$6 # file spec or choice string

# calculates er mapper page dimensions in inches
PAGE_WIDTH=`echo $CANVASWIDTH $DPIX | awk '{printf
"%f", $1/$2}' `
PAGE_HEIGHT=`echo $CANVASHEIGHT $DPIY | awk '{printf
"%f", $1/$2}' `

# sets up temporary plot and postscript files
PLOT_FILE="/tmp/ermapper$$.plt"
PS_FILE="/tmp/ermapper$$.ps" # on both local and
remote machine

# gets the arc/info workspace from the site.erm file
ARC_WS=`$ERMBIN/erm_menu/erm_qsite ARC_WORKSPACE`
if [ $? -ne 0 ]
then

```



```

# prints error message if workspace can't be found
    echo "The ARC/INFO Workspace to use must be defined
in site.erm" 1>&2
    exit 1
fi
# changes workspace name to lower case
ARC_WS=`echo $ARC_WS | tr "[A-Z]" "[a-z]"`

# gets the path to ARC/INFO from the site.erm file
ARC_ARC=`$ERMBIN/erm_menu/erm_qsite ARC_ARC`
if [ $? -ne 0 ]
then
# prints error message if path is not found
    echo "The ARC/INFO program name and path must be
defined in site.erm" 1>&2
    exit 1
fi

# gets remote host from site.erm file
ARC_REMOTE=`$ERMBIN/erm_menu/erm_qsite ARC_REMOTE`
if [ ${ARC_REMOTE:-"LOCAL"} != "LOCAL" ]
then
# code for remote ARC/INFO system
# remote machine is assumed to be a Unix system
# pipe the following to the remote host
cat <<-EOF | rsh $ARC_REMOTE $ARC_ARC 1>&2
    WORKSPACE $ARC_WS # specify workplace
    ARC_PLOT      # block of ARC/PLOT commands
    DISPLAY 1039 1 # specify display
    $PLOT_FILE    # specify name of output plot file
    MAPEXTENT $TLX $BRY $BRX $TLY# specify ER Mapper
map extents
    PAGESIZE $PAGE_WIDTH $PAGE_HEIGHT# specify page
size in inches
    ARCS $FILESPEC # specify cover chosen by user -
arg 15
    QUIT
    POSTSCRIPT $PLOT_FILE $PS_FILE 1 #convert the
plotfile to postscript
    QUIT
    EOF                # end of remote commands

    output_header      # write PS wrapper. set postscript
scaling.
    rsh $ARC_REMOTE cat $PS_FILE      # write the
ARC/INFO PS file to stdout
    rsh $ARC_REMOTE rm -f $PLOT_FILE 1>&2# delete
temporary plot file
    rsh $ARC_REMOTE rm -f $PS_FILE 1>&2# and postscript
file

```

```

        exit 0
    else
        $ARC_ARC <<-EOF 1>&2
        WORKSPACE $ARC_WS # the same procedure for
ARC/INFO
        ARC PLOT          # running locally
        DISPLAY 1039 1
        $PLOT_FILE
        MAPEXTENT $TLX $BRY $BRX $TLY
        PAGESIZE $PAGE_WIDTH $PAGE_HEIGHT
        ARCS $FILESPEC
        QUIT
        POSTSCRIPT $PLOT_FILE $PS_FILE 1
        QUIT
    EOF
    rm -f $PLOT_FILE
    if [ -f $PS_FILE ]
    then
        output_header# except that this checks if
ARC/INFO
        cat $PS_FILE# succeeded in generating the
postscript files
        rm -f $PS_FILE
        exit 0
    else
        echo "ermps_arc: Unable to create the postscript
file." 1>&2
        exit 1
    fi
fi
exit 0

```

# PostScript

PostScript is traditionally a page layout based language, with dimensions specified in points (where there are 72 points to the inch). Dedicated to accurate image processing, ER Mapper is concerned with displaying images to maximum device resolution. The way in which ER Mapper interprets PostScript has been tailored to support this priority. Thus, the key to expressing dimensions in ER Mapper is the following.

## Dimensions in pixels

Dimensions in PostScript code are interpreted by ER Mapper to be in pixels or dots not points.

Thus, the PostScript command `"300 250 rmoveto"` means "move 300 pixels to the right and 250 pixels up".

## Variable canvas size

A second difference between page based applications and ER Mapper is that, for ER Mapper, the canvas size depends on the current size of the image window (or required hardcopy dimensions). For example, a canvas rendered on a 44 x 96 by 400 dpi electrostatic plotter would be 17600 x 38400 pixels, and a canvas for a display window of 4 x 4 at 83 dpi would be 332 x 332 pixels.

## Defined variables

To cope with these varying dimensions, ER Mapper defines four variables for use within its PostScript. These variables are always available in any Dynamic Link. They are:

- `WidthI` - the value of argument 11, the canvas width in pixels
- `HeightI` - the value of argument 12, the canvas height in pixels
- `dpiX` - the value of argument 13, the x device resolution in dots per inch
- `dpiY` - the value of argument 14, the y device resolution in dots per inch



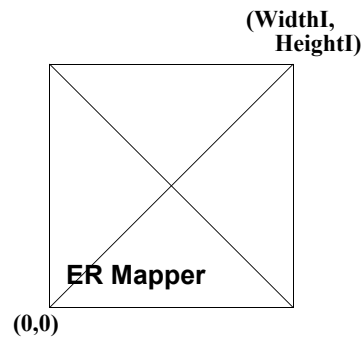
---

where the arguments are those defined in the ["Link initialization"](#) section.

## Example

The example below shows a portion of the code from the PostScript generation program in "Example 1 - Example user dynamic link" above, with the addition of "ER Mapper".

```
0 0 moveto
WidthI 0 lineto
WidthI HeightI lineto
0 HeightI lineto
0 0 lineto
stroke
0 0 moveto
WidthI HeightI lineto
stroke
WidthI 0 moveto
0 HeightI lineto
stroke
/Helvetica-Bold findfont dpiX scalefont setfont
WidthI 0.1 div HeightI 0.1 div moveto
(ER Mapper) show
showpage
```



**PostScript code making use of the ER Mapper defined variables**



*The value of the "WidthI" and "HeightI" dimensions change with the size of the canvas, whereas the device resolution variable "dpiX" has been used to specify text exactly 1 inch high, regardless of the size of the canvas.*

## ER Mapper PostScript

Dynamic Link generation programs must take into account ER Mapper's pixel coordinate system. In addition, pixel-based PostScript files can be displayed using ER Mapper's "Dynamic Link coded PostScript" "Monocolor" and "Truecolor" Dynamic Links.

## Standard Postscript

PostScript from other sources must be scaled from a "72 points per inch" to a "pixels" coordinate system.

For example, the following code, from ER Mapper's "8.5x11 inch Monocolor PostScript" Dynamic Link, scales PostScript files to fit the image window. The files are assumed to have been created using a standard drawing or writing package on an 8.5x11 inch page.

The variables "WidthI" and "HeightI" are the canvas width and height passed to the Dynamic Link in parameters 11 and 12 as defined on the previous page. For example, in the following PostScript:

```
/inch {72 mul} def WidthI 8.5 inch div HeightI 11.0 inch
div scale
```

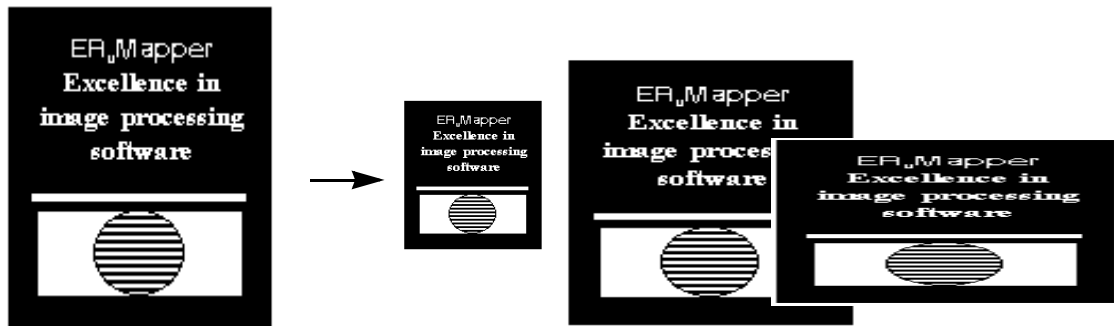
'WidthI 8.5 inch div' is the x scale factor and 'HeightI 11.0 inch div' is the y scale factor.



*If your image is composed on a different size page to 8.5 x 11 inch the overlay won't fit the image exactly. Similar Dynamic Links can easily be constructed to scale other sizes of PostScript page (for example A4, or 3 inch x 5 inch) to the ER Mapper canvas.*

### Variable aspect ratio

The figure below shows how this scaling squashes the image to fit in the current window, varying the image aspect.



### Dimensions in points

There are two remaining ER Mapper defined variables which may be useful, especially if you have a printing background and are used to working in points. These are:

`from72pt` converts dimensions in points to dimensions in pixels. It is defined as:

```
/from72pt {dpiX 72 div mul} def
```

Thus, for example, you would use:

```
12 from72pt scalefont - to set a font to 12 points, 72 from72pt
72 from72pt rlineto - to move right 1 inch and up 1 inch.
```

Similarly,

`fromdevpt` converts dimensions in pixels to dimensions in 72 point. It is defined as:

```
/fromdevpt { 72 dpiX div mul} def
```

## Summary of ER Mapper PostScript functions

<b>Specifying dimensions and position</b>		
Dimension type	Usage	examples
ABSOLUTE Dimensions in points	<ul style="list-style-type: none"> <li>Useful for setting absolute text sizes or line thicknesses</li> <li>Use ER Mapper defined procedure "from72pt"</li> </ul>	8 from72pt setlinewidth 12 from72pt scalefont
ABSOLUTE Dimensions in inches	<ul style="list-style-type: none"> <li>Useful for specifying absolute dimensions in inches</li> <li>Use ER Mapper defined variables "dpiX" and "dpiY"</li> </ul>	3 dpiX mul 2 dpiY mul rlineto dpiX 2 div setlinewidth
RELATIVE Dimensions in pixels	<ul style="list-style-type: none"> <li>Useful for defining position</li> <li>Use corner coordinates, for example the shell script "TLX", "TLY", "BRX" and "BRY", and canvas dimensions "WidthI" and "HeightI" to scale image coordinates to PostScript pixel coordinates</li> <li>Use WidthI and HeightI to specify dimensions relative to the window</li> <li>The dimensions specified in pixels depend on the current device resolution</li> </ul>	<pre> /x E \${TLX} sub WidthI mul \${BRX} \${TLX} sub div  0.1 WidthI 0.9 HeightI moveto 0.05 HeightI setlinewidth 10 setlinewidth           </pre>

<b>Global ER Mapper defined PostScript variables</b>		
variable/procedure name	Dynamic Link parameter number/procedure definition	explanation

WidthI	• argument 11	canvas width in pixels
HeightI	• argument 12	canvas height in pixels
dpiX	• argument 13	device resolution in dots per inch X
dpiY	• argument 14	device resolution in dots per inch Y
from72pt	• dpiX 72 div mul	scale by dpiX/72
fromdevpt	• 72 dpiX div mul	scale by 72/dpiX

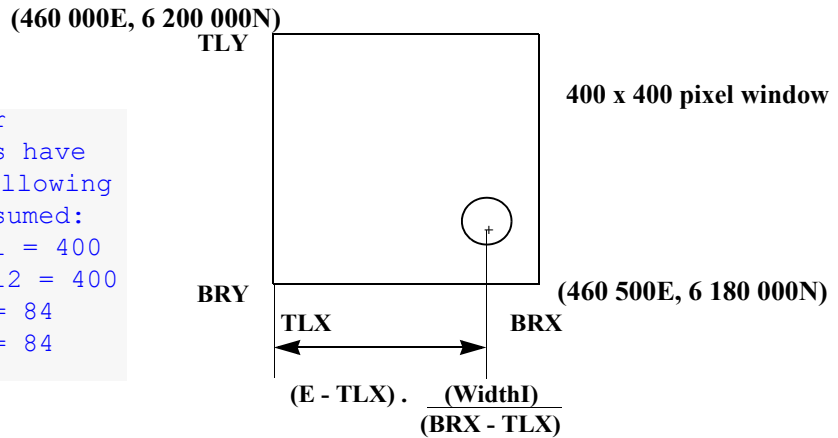
### Example

The example below is based on the "Table of Data Shown as Circles" Dynamic Link . It only draws a single circle while the link can process a table of eastings, northings and values. The code for the entire Dynamic Link is given in the "[Displaying the data](#)" section above.

<b>%!</b>	<b>start of PostScript</b>
/TLX 460000 def /TLY 6200000 def	defines the top left coordinate
/BRX 460500 def /BRY 6180000 def	defines the bottom right coordinate
/N 6185000 def	defines the variable N (Northing) with value 6185000
/E 460400 def	defines the variable E (Easting) with value 460100
/R dpiX 2 div def	sets the radius to 1/2 inch
/x_scale {WidthI BRX TLX sub div} def	defines $x\_scale = WidthI / (BRX - TLX)$
/y_scale {HeightI TLY BRY sub div} def	defines $y\_scale = HeightI / (TLY - BRY)$
gsave	stores the current origin position
E TLX sub x_scale mul	converts the easting value to position across the screen in terms of pixels and leaves this on the stack
N BRY sub y_scale mul	converts the northing value to position up the screen in terms of pixels and adds this to the stack
translate	the point defined by the two stack entries becomes the new origin
0 0 R 0 360 arc stroke	draws a circle centered on the new origin
-2 -2 moveto (+) show	draws the center of the circle

grestore	returns to the previous origin
showpage	sends the page to the output device

The global ER Mapper PostScript variables have been used and the following values have been assumed:  
WidthI = argument 11 = 400  
HeightI = argument 12 = 400  
dpiX = argument 13 = 84  
dpiY = argument 14 = 84



The radius is set absolutely at 1/2 inch using the code `"/R dpiX 2 div"` where `dpiX` is the number of pixels in one inch for the current device. In the same way an absolute dimension of three inches would be specified as `"3 dpiX mul"` and so on.

The pixel coordinates of the target point, specified in Easting-Northing coordinates (E,N), are calculated using the EN extents of the current window and the pixel dimensions of the window.

## Performance and accuracy

Your PostScript can have a significant impact on the performance and accuracy of your Dynamic Link. A well constructed Dynamic Link has high performance and high accuracy (equal to or greater than the DPI of the output device). **Remember: ER Mapper Dynamic Links are always generated at device resolution.**

Performance and accuracy will be maximized if you pay attention to the following points.

### Use device DPI

Work at device DPI where possible to ensure maximum accuracy.

### Integers not reals

Use integers scaled to the device DPI. PostScript processes integers faster than reals and there is little point in drawing at higher resolution than the device DPI.

### Reduce PostScript file size

Use `"defines"` to reduce your PostScript text file size as this will reduce the processing time. In the example below, the second column of code is faster than the first because it has fewer characters and uses integer rather than real values.



## Use Monocolour PostScript

Use MONOCOLOR Dynamic Links instead of TRUECOLOR Dynamic Links unless you intend your output to create multi-color overlays. MONOCOLOR links consume less memory when running and are faster.

## Extract current data

For Dynamic Links to GIS, DBMS and other such systems, try to extract data for only the area currently being covered by the ER Mapper image. There is no point in extracting data outside the image window (although there is no harm in doing so) and the less data being processed by the link, the faster it will run.

## Use ER Mapper coordinates

Remember ER Mapper PostScript is based on device pixels, not points. Your dimensioning commands, such as `scalefont`, `setlinewidth`, `moveto` and `lineto` calls, need to take this into account.

For example, consider the following two example Dynamic Links, both of which draw vectors:

Example of slow PostScript	Example of fast PostScript
<pre>1 from72pt setlinewidth 0.0 0.0 moveto 100.1 200.7 lineto 300.3 400.9 lineto 100.3 400.9 lineto 200.3 400.9 lineto 300.3 400.9 lineto stroke</pre>	<pre>/W {from72pt setlinewidth} def /M moveto def /L lineto def /S stroke 1 W 0 0 M 100 201 L 300 401 L 100 401 L 200 401 L 300 401 L S</pre>

## Using existing PostScript

Many systems to which you may wish to create a Dynamic Link support PostScript output. All that is necessary to use this output is to rescale it to the size of the ER Mapper image.

For example, suppose you have a system you wish to connect to which always outputs PostScript scaled such that (0.0, 0.0) is bottom-left and (2.0, 4.0) is top-right of the image.

To rescale this PostScript so that ER Mapper will use it, do:

```
WidthI 2.0 div HeightI 4.0 div scale% scale to ER
Mapper size
```

## ervecmacro.erm

The 'ervecmacro.erm' file can be found in the 'config' directory. It is a file used internally by ER Mapper to convert ERVEC files into PostScript format before output to a screen or printer. It is a Read Only file and must not be altered.



# The ER Mapper Configuration File (config.erm)

The configuration file provides information to ER Mapper about the computer environment in which it is operating. For Unix installations, the hardware configuration (display types etc.) to be used on the network and software license information for each computer running ER Mapper are stored in 'config.erm'.

The various selections for each entry in the configuration file are discussed in detail below. The example included in this chapter contains entries for a typical multi-license site.

## ERMConfig Block

The configuration data is contained in a block in a similar manner to other ER Mapper ascii text information files. The block name ERMConfig encloses the entries for the configuration. Individual entries are described below.

- **Version:** The version of 'config.erm' definition.

For example,

```
Version = "5.0"
```

- **PostScriptPrinter:** The command to output to your PostScript printer, if you have one. For example:

```
PostScriptPrinter="lpr -Ppostscript"
```

The shipped version of ER Mapper has:

```
PostScriptPrinter = "lpr"
```

If you have a printer, such as a HP laserjet, that can print line output, but not PostScript, you should comment out the PostScriptPrinter entry to stop ER Mapper from thinking it has access to a PostScript printer.

- **LinePrinter:** The command to output to your line printer, if you have one. For example, ER Mapper is shipped with the following line:

```
LinePrinter = "lpr"
```

- **Units:** The units entry specifies the unit type for all quantities displayed by ER Mapper. The units must be expressed in metric. For example,

```
Units = Metric
```

- **DefaultRGLUT:** Not used.
- **DefaultHSILUT:** Not used.

- **DefaultRtShade:** Not used.
- **DefaultHardcopy:** The default device for printing hardcopy images. If more than one hardcopy device is available on the network, the default device may be a fast, low resolution printer for rapid assessment of a scene. For example,

```
DefaultHardcopy = "PaintJet"
```

- **DefaultHistStyle:** The default histogram style in the **Transform** window.

There are three allowed values for the DefaultHistStyle:

- Non-Cumulative - Histogram which is a visual representation of the relative frequency for each discrete data value.
- Cumulative - Histogram showing the number of data values that are below the upper end point of each data value.
- DeQuantised - Histogram showing the relative frequency for a continuous set of data values. For example, data with floating point type data values would be better displayed as a DeQuantised histogram.

For example,

```
DefaultHistStyle = DeQuantised
```

- **EnforceAspectRatio:** If the aspect ratio is enforced the pixel size and pixel overlap information for the image files being processed is used to produce an image in an ER Mapper window which has the correct aspect ratio. The pixel distance in the horizontal and vertical axis is always forced to 1:1. The following example entry in the configuration file enforces a 1:1 aspect ratio:

```
EnforceAspectRatio = Yes
```

This option may be changed using the **Edit/Preferences** command.

- **DoOptimisation:** Enables operator to specify whether or not to optimise processing for speed. Usually would be set to 'yes'. For example, in the configuration file the following entry would appear:

```
DoOptimisation = Yes
```

- **CLibrarySwitches:** Default Id switches to use when loading user C object code into ER Mapper.

# Algorithm files (.alg)

Algorithms define the images and processing to use to generate a particular output image. Because they are stored separately from the data they can easily be edited and used with other similar images.

You don't need to type in a text "algorithm file" - you use ER Mapper's Graphical User Interface to interactively define the processing you want and when you save the algorithm ER Mapper creates or edits the algorithm file automatically.

Algorithm files define the data sources, filters, transformations and formulae used to create the desired image.

The algorithm file format is subject to change.

## Example algorithm file

The following algorithm file, was created by simply clicking on the necessary processing steps within the Algorithm window and then saving it to disk with the file name "examples\Applications\Land\_Information\two\_surface\_example.alg". In this example, the algorithm includes two surfaces, one with a pseudocolor layer and vector layer, and the other with a pseudocolor and intensity layer.

```
Algorithm Begin
  Version= "5.7"
  Name = "two_surface_example"
  Description= "Example algorithm with two surfaces"
  Author= "ER Mapper"
  Comments= "Example two surface algorithm"
  ViewMode= Perspective3D
  LastUpdated= Thu Sep 03 01:48:32 GMT 1998
  BackgroundColourSet= No
  BackgroundColour= 0,0,0
  ConvertToGreyscale= No
  CoordinateSpace Begin
    Datum = "NAD27"
    Projection= "NUTM11"
    CoordinateType= EN
    Rotation= 0:0:0.0
  CoordinateSpace End
  TopLeftCorner Begin
    Eastings= 505564.9746060915
    Northings= 3651045.652393905
  TopLeftCorner End
  BottomRightCorner Begin
    Eastings= 525263.3836848846
    Northings= 3631347.243315111
  BottomRightCorner End
  MosaicType= Overlay
  PageSize Begin
```

```

PageConstraint= Scale
PageWidth= 8.5
PageHeight= 11
TopBorder= 0.3937007874015748
BottomBorder= 2.419067768883412
LeftBorder= 0.3937007874015748
RightBorder= 0.4094488188976378
Scale = 70502.53939080733
PageExtents Begin
  TopLeftCorner Begin
    Eastings= 505564.9746060919
    Northings= 3651045.652393908
  TopLeftCorner End
  BottomRightCorner Begin
    Eastings= 520786.4724096644
    Northings= 3631347.243315116
  BottomRightCorner End
PageExtents End
ContentExtents Begin
  TopLeftCorner Begin
    Eastings= 506270
    Northings= 3650340.627
  TopLeftCorner End
  BottomRightCorner Begin
    Eastings= 520053.246
    Northings= 3635679.224
  BottomRightCorner End
ContentExtents End
PageSize End
ThreeDInfo Begin
  DrawMode= Colorfill
  WindowAspectRatio= 1
  ViewMatrix= {
    1 0 0 0
    0 1 0 0
    0 0 1 0
    0 0 0 1
  }
  ModelMatrix= {
    0.96592581272130.06698727607727-
0.249999994039540
    -0.2588190138340.2499999701977-
0.93301266431810
    0 0.96592581272130.2588190138340
    0 0 0 1
  }
  HeadMatrix= {
    1 0 0 0
    0 1 0 0
    0 0 1 0

```

```

        0 0 0 1
    }
    ScreenAxesRotn= Yes
    Velocity= 0.099999999403953552
    TurnRate= 9.999999747378752E-005
    EyeXYZ= {
        0 -250
    }
    Left = -47
    Right = 47
    Bottom= -47
    Top = 47
    Near = -188
    Far = 103.4000015258789
    FlyNear= 1
    FlyFar= 400
    FlyFOV= 45
    MaterialAmbient= {
        0 0 0.50196075439451
    }
    MaterialDiffuse= {
        1 1 1 1
    }
    MaterialSpecular= {
        0.89999991655350.89999991655350.8999999165535
1
    }
    MaterialShininess= 0
ThreeDInfo End
Surface Begin
    Name = "Default Surface"
    ZOffset= 0
    Transparency= 0
    ColorMode= PSEUDO
    LookupTable= "pseudocolor"
    Stream Begin
        Type = DynamicLink
        Description= "100m vector contours"
        Dataset= "Muth_Valley_100m_contours.erv"
        FilterProgramName= "ermps_erverc"
        LinkType= TrueColour
        FileExtent= ".erv $$ALG"
        CanEdit= Yes
        EditProgram= "erm_vec"
        RGBcolour Begin
            Red = 0
            Green = 0
            Blue = 0
        RGBcolour End
    Stream End

```

```

Stream Begin
  Type = Pseudo
  Description= "100m color contours"
  Dataset=
  ".../Shared_Data/Digital_Terrain_Model_20m.ers"
  DoStaticShade= No
  SunAzimuth= 45:0:0.0
  SunAngle= 45:0:0.0
  StreamInput Begin
    StreamInputID= 1
    Kernel Begin
      Type = Convolution
      Directory= "filters_lowpass"
      Name = "avg5"
      Description= "5x5 Average filter"
      Rows = 5
      Columns= 5
      OKOnSubsampledData= Yes
      Array = {
        1 1 1 1 1
        1 1 1 1 1
        1 1 1 1 1
        1 1 1 1 1
        1 1 1 1 1
      }
      ScaleFactor= 25
    Kernel End
  StreamInput End
  Formula Begin
    Directory= "contour"
    Name = "contour_20_unit_interval"
    Description= "Contour with 20 unit gap"
    Formula= "CEIL( (I1-rmin(,r1,i1))/100+1)"
    FormulaArg Begin
      StreamInput= 1
      BandNumber= 1
      BandId Begin
        Value = "Pseudo"
      BandId End
    FormulaArg End
    RegionArg Begin
      RegionNumber= 1
      RegionName= "All"
    RegionArg End
  Formula End
  Transform Begin
    Type = Linear
    MinimumInputValue= 1
    MaximumInputValue= 12
    MinimumOutputValue= 0

```



```

MaximumOutputValue= 255
LinearDef Begin
  NumberOfPoints= 2
  Points= {
    0.0000000.000000
    1.0000001.000000
  }
LinearDef End
Histogram Begin
  MinimumValue= 3
  MaximumValue= 12
  CellsBelowMin= 0
  CellsAboveMax= 0
  NrOfCells= 12160
  Bins = {
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 15440 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    19600 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 18630 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 40070 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 1462
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 7240 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 2800 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 214
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 92 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 14
  }
Histogram End
DoInRealtime= No
Transform End
Stream End
Surface End
Surface Begin
  Name = "Default Surface"
  ZScale= 772.4866305408812

```

```

ZOffset= -127.5
Transparency= 0
ColorMode= PSEUDO
LookupTable= "waterlevel"
Stream Begin
  Type = Intensity
  Description= "Intensity Layer"
  Dataset= "Landsat_MSS_with_100m_contours.alg"
  DoStaticShade= No
  SunAzimuth= 45:0:0.0
  SunAngle= 45:0:0.0
  StreamInput Begin
    StreamInputID= 1
  StreamInput End
  Formula Begin
    Description= "Default Formula"
    Formula= "INPUT1"
    FormulaArg Begin
      StreamInput= 1
      BandNumber= 3
      BandId Begin
        Value = "Red"
      BandId End
    FormulaArg End
  Formula End
  Transform Begin
    Type = Linear
    MinimumInputValue= 0
    MaximumInputValue= 255
    MinimumOutputValue= 0
    MaximumOutputValue= 255
    LinearDef Begin
      NumberOfPoints= 2
      Points= {
        0.0000000.000000
        1.0000001.000000
      }
    LinearDef End
    Histogram Begin
      MinimumValue= 0
      MaximumValue= 255
      CellsBelowMin= 0
      CellsAboveMax= 0
      NrOfCells= 13447
      Bins = {
        1650 0 0 0 0 95 0 0 0
        0 0 0 47 0 0 0 0 0
        30 0 0 0 0 0 0 0 30 0
        0 0 0 0 0 40 0 0 0 0
        0 0 81 0 0 0 0 0 0 104
      }
    Histogram End
  Transform End

```

```

0 0 0 0 0 0 1910 0 0
0 0 0 2420 0 0 0 0 0
2720 0 0 0 0 0 0 3590
0 0 0 0 0 5050 0 0 0
0 0 6230 0 0 0 0 0 779
0 0 0 0 0 0 8560 0 0
0 0 0 8700 0 0 0 0 0
0 9340 0 0 0 0 0 10030
0 0 0 0 0 9530 0 0 0
0 0 8210 0 0 0 0 0 697
0 0 0 0 0 0 6070 0 0
0 0 0 4470 0 0 0 0 0
0 4230 0 0 0 0 0 3430
0 0 0 0 0 3720 0 0 0
0 0 3230 0 0 0 0 0 285
0 0 0 0 0 0 2190 0 0
0 0 0 0 1650 0 0 0 0
0 1120 0 0 0 0 0 99 0
0 0 0 0 0 77 0 0 0 0
0 0 57 0 0 0 0 0 0 46
0 0 0 0 0 175
}
Histogram End
DoInRealtime= No
Transform End
Stream End
Stream Begin
Type = Pseudo
Description= "Pseudo Layer"
Dataset= "Landsat_MSS_with_100m_contours.alg"
DoStaticShade= No
SunAzimuth= 45:0:0.0
SunAngle= 45:0:0.0
StreamInput Begin
StreamInputID= 1
StreamInput End
Formula Begin
Description= "Default Formula"
Formula= "INPUT1"
FormulaArg Begin
StreamInput= 1
BandNumber= 1
BandId Begin
Value = "Blue"
BandId End
FormulaArg End
Formula End
Transform Begin
Type = Linear
MinimumInputValue= 0

```

```

MaximumInputValue= 255
MinimumOutputValue= 0
MaximumOutputValue= 255
LinearDef Begin
  NumberOfPoints= 2
  Points= {
    0.0000000.000000
    1.0000001.000000
  }
LinearDef End
Histogram Begin
  MinimumValue= 0
  MaximumValue= 255
  CellsBelowMin= 0
  CellsAboveMax= 0
  NrOfCells= 13452
  Bins = {
    5230 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 11070 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 18200 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    20870 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 17630 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 14090 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 11030 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    9360 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 7540 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 5730 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 3860 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 278
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 2100 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 1740 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 1160 0 0 0 0 0 0 0
    0 0 0 0 0 213
  }
Histogram End
DoInRealtime= No
Transform End
Stream End
Surface End
Algorithm End

```

See the algorithms in the subdirectories within the 'examples' directory for other examples.



The supplied algorithms are listed in the [Supplied Algorithms](#) chapter in the *ER Mapper Applications manual*.

## Algorithm file entries

The entries in the Algorithm file are listed below. They consist of some overall settings followed by blocks defining the processing.

- **Version:** The version number the algorithm was last saved in. This should agree with the release of ER Mapper software you are currently using.

```
Version = "6.0"
```

- **Name:** Only used internally by ER Mapper.
- **Description:** A short description of the Algorithm to be displayed in the layer in the Algorithm dialog. This can be edited in the Algorithm window. For example,

```
Description = "Example algorithm with two surfaces"
```

- **Author:** The person who created the algorithm. For reference only - not currently used. For example,

```
Author = "ER Mapper"
```

- **Comments:** A description of the algorithm. It can contain multiple lines of text. This field can be viewed when loading an algorithm by pressing the **Comments** button.

- **PageViewMode:** Describes the current algorithm Page View Mode setting. If the algorithm has Page setup parameters set you can specify whether the image is to be displayed on its own (normal) or as part of the page layout (layout). If omitted, this entry defaults to normal. This is only applicable to 2D. Allowable PageViewMode values are:

- normal
- layout

For example,

```
PageViewMode = layout
```

- **ViewMode:** Describes the current algorithm View Mode setting. Allowable ViewMode values are:

- Planimetric2D
- Perspective3D
- Flythrough

For example,

```
ViewMode = Planimetric2D
```

- **LastUpdated:** Date the algorithm was last written to disk. For example,

```
LastUpdated = Mon Aug 31 07:25:15 GMT 1998
```

- **BackGroundColourSet:** Indicates whether the background color is set in the algorithm. Allowable values are "Yes" and "No". For example,

```
BackGroundColorSet = No
```

- **BackGroundColor:** The Red,Green,Blue values specified in the algorithm. For example,

```
BackGroundColor = 0,0,0
```

- **ConvertToGreyscale:** Indicates whether the background color is converted to greyscale. Allowable values are "Yes" and "No". For example,

```
ConvertToGreyscale = No
```

- **CoordinateSpace Block:** Defines the coordinate space of the algorithm. See ["Coordinate Space block"](#) below.

- **TopLeftCorner Block, BottomRightCorner Block:** The coordinates of the top left corner of the algorithm extents. See ["TopLeftCorner and BottomRightCorner blocks"](#) below.

- **SuperSampling:** Indicates whether or not smoothing is desired. This is set using the Smoothing checkbox in the Algorithm dialog. Possible values are:

- Bilinear - Smoothing is selected
- NearestNeighbour - Smoothing is not selected

For example,

```
SuperSampling = Bilinear
```

- **MosaicType:** In algorithms with different images loaded into streams of the same type, `MosaicType` specifies how the edge where the images overlap should appear. Can be either:

- `Overlay` - where an image overlaps another the top one is shown
- `Feather` - where an image overlaps another values are interpolated across the overlap to give a smooth join.

For example,

```
MosaicType = Overlay
```

- **PageSize Block:** Defines the page layout parameters for the algorithm. See ["PageSize block"](#) below.
- **ThreeDInfo Block:** Defines the 3D viewing characteristics of the algorithm including images and processing. See ["ThreeDInfo block"](#) below.
- **Surface Block:** Defines the characteristics of the algorithm surface, including images and processing. There is one block for each surface on the algorithm. See ["Surface blocks"](#) below.

## Coordinate space block

This describes the coordinate space that all image datasets in the algorithm must be compatible with. This is maintained automatically. If there is only one image in an algorithm and the user changes it to an image with a different coordinate space, the algorithm coordinate space will be changed.

```
CoordinateSpace Begin
Datum = "AGD66"
Projection = "TMAMG54"
CoordinateType = EN
Units = "METERS"
Rotation = 0:0:0.0
CoordinateSpace End
```



*The Units entry does not apply to the cell-sizes shown in theCellInfo block.*

The information within the `CoordinateSpace` block is read from the `.ers` dataset header and contains the following entry lines.

- **Datum:** Defines the geodetic datum of the map projection. Allowable types are RAW or any of the datums supported by ER Mapper.

For example,

```
Datum = AGD66
```

- **Projection:** Defines the map projection; Allowable values are RAW or one of the map projection supported by ER Mapper.

For example,

```
Projection = TMAMG54
```

- **CoordinateType:** The CoordinateType defines how the coordinates are expressed. Allowable values are:

- RAW,
- LL and
- EN.

For example,

```
CoordinateType = EN
```

- **Units:** (Optional). Specifies the units of length. The most common units are "METERS" and "natural". If the units are not specified, they default to "METERS" for RAW images and "natural" for non-RAW images.
- **Rotation:** Defines the rotation of the image from true North. For example,

```
Rotation = 0:0:0.0
```

## TopLeftCorner and BottomRightCorner Blocks

These are the coordinates of the top left and bottom right corners of the image expressed in the coordinates defined by the coordinate space. In this case the top left corner represents cell (0,0) of the "Newcastle" image. The coordinates are expressed in the CoordinateType specified in the CoordinateSpace Block (above). In this case, in Eastings/Northings.

```
TopLeftCorner Begin
  Eastings = 775700
  Northings = 7997200
TopLeftCorner End
BottomRightCorner Begin
  Eastings = 827300
  Northings = 7930000
BottomRightCorner End
```

## PageSize Block

Page size block stores information about the preferred page size, border widths and image extents.



For example,

```
PageSize Begin
  PageConstraint= Scale
  PageWidth = 8.5
  PageHeight = 11
  TopBorder = 0.3937007874015748
  BottomBorder = 2.419067768883412
  LeftBorder = 0.3937007874015748
  RightBorder = 0.4094488188976378
  Scale = 70502.53939080733
PageExtents Begin
  TopLeftCorner Begin
    Eastings = 505564.9746060919
    Northings = 3651045.652393908
  TopLeftCorner End
  BottomRightCorner Begin
    Eastings = 520786.4724096644
    Northings = 3631347.243315116
  BottomRightCorner End
PageExtents End
ContentExtents Begin
  TopLeftCorner Begin
    Eastings = 506270
    Northings = 3650340.627
  TopLeftCorner End
  BottomRightCorner Begin
    Eastings = 520053.246
    Northings = 3635679.224
  BottomRightCorner End
ContentExtents End
PageSize End
```

- **PageConstraint:** The type of page constraint for the algorithm. Allowable values are:
  - None - the page contents are set by the current image window zoom
  - Page - the page size varies automatically
  - Border - the border varies automatically
  - Scale - the scale varies automatically

For example,

```
PageConstraint = Scale
```

- **PageWidth:** The width of the page in inches. For example,

PageWidth = 8.5

- **PageHeight:** The height of the page in inches. For example,

PageHeight = 11

- **TopBorder, BottomBorder:** The size of the top and bottom borders in inches. For example,

TopBorder = 0.3937007874015748

BottomBorder = 2.419067768883412

- **LeftBorder, RightBorder:** The size of the left and right borders in inches. For example,

LeftBorder = 0

RightBorder = 0

- **Scale:** The scale of the image compared to ground units. For example,

Scale = 4182.491894396

- **PageExtents Sub-Block:** The coordinates of the top left and bottom right corners of the complete page. Specified in the same way as the "*TopLeftCorner and BottomRightCorner blocks*" above.

- **ContentExtents Sub-Block:** The coordinates of the top left and bottom right corners of the image area on the page. Specified in the same way as the "*TopLeftCorner and BottomRightCorner blocks*" above.

## ThreeDInfo block

The ThreeDinfo block contains the three dimensional viewing parameters of the algorithm.

- **DrawMode:** Sets the 3D mode in which the image is displayed. Allowable values are:

- WireframeHV

- Colorfill

- Textured

- Auto

For example:

DrawMode = Colorfill

- **WindowAspectRatio:** Sets the aspect ratio of the image window when viewing the image in 3D. A square window has an aspect ratio of 1.
- **ViewMatrix, ModelMatrix, HeadMatrix:** These matrixes set the position of the 3D image. ModelMatrix is used for 3D perspective and ViewMatrix and HeadMatrix are used for 3D flythrough.

The following example shows the matrixes when viewing the image from the top.

```

ViewMatrix= {
    1    0    0    0
    0    1    0    0
    0    0    1    0
    0    0    0    1
}
ModelMatrix= {
    1    0    0    0
    0    1    0    0
    0    0    1    0
    0    0    0    1
}
HeadMatrix= {
           1    0    0    0
    0    1    0    0
    0    0    1    0
    0    0    0    1
}

```

- **DisplayBBox:** If set to Yes, it draws a box around the image. If this entry is omitted, it defaults to No. For example:

```
DisplayBBox = Yes
```

- **AllLightsOff:** Disables artificial lighting for 3D viewing if set to Yes. If this entry is omitted, it defaults to No. For example:

```
AllLightsOff = Yes
```

- **ScreenAxesRotn:** If set to Yes, indicates that the image can be rotated on the screen. ER mapper will automatically change this to Yes if you view the image in 3D.
- **Velocity, TurnRate:** These indicate the speed of movement in 3D Flythrough.
- **EyeXYZ:** Sets the coordinates of the position from which the image is viewed in 3D. For example:

```

EyeXYZ = {
    0  -25  0
}

```

- **Left, Right, Bottom, Top, Near, Far:** Sets the extents of the 3D image. For Example:

```
Left = -47
Right = 47
Bottom = -47
Top = 47
Near = -188
Far = 103.4000015258789
```

- **FlyNear, FlyFar, FlyFOV:** Sets the position and field of view of the image when viewed in 3D Flythrough. For example:

```
FlyNear = 1
FlyFar = 400
FlyFOV = 45
```

- **MaterialAmbient:** Sets the color of the overall lighting illuminating the image in 3D. For example:

```
MaterialAmbient = {
0.5 0.5 0.5
}
```

- **MaterialDiffuse:** Sets the color of the lights directly illuminating the image in 3D. For example:

```
MaterialDiffuse = {
1 1 1 1
}
```

- **MaterialSpecular:** Sets the color of the highlights of the image in 3D. For example:

```
MaterialSpecular= {
0.89999999761581 0.899999997615810.899999997615811
}
```

- **MaterialShininess:** Sets the amount of shine in the 3D image. This ranges from 0 for no shininess to 100 for full shininess. For example:

```
MaterialShininess = 50
```

## Surface Blocks

Each surface in the algorithm is represented by Surface Block.

- **Name:** The name of the surface as it appears on the algorithm dialog. This defaults to "Default Surface" if it is not specified. For example:

```
Name = "Surface 1"
```

- **ZScale:** Sets the height exaggeration as a percentage of the x and y measurements of the surface; i.e a setting of 100 will draw the height to the correct scale. For example:

```
ZScale = 772.5
```

- **ZOffset:** Sets the vertical position of the surface in relation to an arbitrary zero base point in the same units as the x and y units. For example:

```
ZOffset = -127.5
```

- **Transparency:** Sets the viewing transparency of the surface as a percentage. A value of 0 represents no transparency and 100 represents full transparency. For example:

```
Transparency = 10
```

- **ColorMode:** Describes the surface color mode. Allowable ColorMode values are:

- RGB
- HSI
- PSEUDO
- RTSHADE
- DATASET - Used for write to image only

For example,

```
ColorMode = PSEUDO
```

- **LookupTable:** Defines the default color lookup table for the surface. When the algorithm is loaded, the lookup table for this surface will be set to this lookup table automatically. For example,

```
LookupTable = "pseudocolor"
```




---

See the [“Look up table files \(.lut\)”](#) chapter of this manual for the format of color look up table files.

- **Stream Block:** Defines the layers in the surface including images and processing. See [“Stream blocks”](#) below.

## Stream Blocks

Stream blocks define the layer information within the surface. There is one stream block for each layer.

All types of layer have the following entries.

- **Type:** The type of layer. Allowable values are:
  - Red
  - Green
  - Blue
  - Hue
  - Saturation
  - Intensity
  - Pseudo
  - DynamicLink
  - Classification
  - Height

For example,

```
Type = Pseudo
```

- **Description:** The layer description as entered in its label in the **Algorithm** dialog box.
- **Dataset:** The directory and filename of the image dataset from which the data is originating for the layer. The directory path is relative to the algorithm directory.

```
Dataset=
"../../Shared_Data/Digital_Terrain_Model_20m.ers"
```




---

*Raster layers also have the following parameters and sub-blocks. The parameters for dynamic link layers are described in ["Dynamic link overlay entries"](#) at the end of this chapter.*

- **DoStaticShade:** This is set to `No` if static shading is disabled.
- **SunAzimuth and SunAngle:** The azimuth and angle used for static shading layers. This would also be used when output is a real time shade overlay to a hardcopy device or to an image. This is adjusted either through the static or realtime shade sun angle selector.
- **StreamInput Sub-blocks:** Defines ...See ["StreamInput sub-blocks"](#) below.
- **Formula Sub-blocks:** See ["Formula sub-blocks"](#) below.

- **Transform Sub-blocks:** See ["Transform sub-blocks"](#) below.

## StreamInput Sub-Blocks

In general, each stream input corresponds to a band from an image, along with optional input filters and transforms. In the example at the beginning of this chapter the Intensity layer has one stream input:

```
StreamInput Begin
  StreamInputID= 1
  Kernel Begin
  .....
  Kernel End
StreamInput End
```

- **StreamInputID:** The identifier for the input. In the formula "INPUT1" will refer to:

```
StreamInputID = 1
```

- **Kernel Blocks:** The Kernel Block defines a filter for the band.

## Kernel Sub-Blocks

Kernel sub-blocks may occur within the StreamInput block. Kernel blocks may also be stored as separate ascii files. The information contained in a kernel file is copied into the algorithm file when a filter (kernel) is selected.



See the ["Filter files \(.ker\)"](#) chapter in this manual for more information.

## Formula Sub-Blocks

Formula blocks contain the formula definition for the overlay. Formula blocks are made up of the following elements:

- **Name:** The Name is displayed in the formula window and available for editing.
- **Description:** Formula description displayed in the formula window and available for editing. For example,

```
Description = "Default Formula"
```

- **Formula:** The actual formula entry. The formula is within double quotes.



Syntax for formulae is described in the [Formula syntax](#) chapter in the ER Mapper User Guide.

For example, the default formula is:

```
Formula = "I1"
```

- **FormulaArg Blocks:** FormulaArg blocks define the band that is associated with each formula input. For example:

```
FormulaArg Begin
  StreamInput= 1
  BandNumber= 1
  BandId Begin
    Value = "magnetics"
  BandId End
FormulaArg End
```

There will be as many FormulaArg blocks as there are inputs for the formula.



See also "[Formula files \(.frm\)](#)" in this manual for more information.

## Transform Sub-Blocks

Transform blocks occur before the Formula block and/or after the Formula block. In the example presented at the beginning of this chapter, there is one Transform blocks after the formula in both the intensity overlay and the pseudocolor overlay.

Transform blocks contain the following elements:

- **Type:** The type of transform for the transform block. Allowable types are:
  - Linear
  - Exponential
  - Logarithmic
  - Histogram Only

For example,

```
Type = Linear
Type = Logarithmic
```

- **MinimumInputValue:** The minimum value of the data.
- **MaximumInputValue:** The maximum value of the data.



- **MinimumOutputValue:** The minimum value of the transformed (output) data.
- **MaximumOutputValue:** The maximum value of the transformed (output) data.
- **Transform Point Coordinate Block:** The name of the transform point coordinate block depends on the type of transform. For example, for a transform of Type Linear, the transform point coordinate block will be named LinearDef. In this block, the number of coordinate points is given as well as the x and y coordinates for each of these points. For example,

```

LinearDef Begin
  NumberOfPoints = 2
  Points = {
    0.0000000.000000
    1.0000001.000000
  }
LinearDef End

```

## Dynamic Link Overlay Entries

An example dynamic link layer is shown below:

```

Stream Begin
  Type = DynamicLink
  DatasetDirectory = "Shared_Data"
  Dataset = "Newcastle_structure.erv"
  MenuButtonDescription = "Annotation overlay"
  FilterProgramName = "ermps_ervec"
  LinkType = Monocolour
  FileExtent = ".erv"
  InitialisationProgram = "ermit_ers"
  CanEdit = Yes
  EditProgram = "erm_vec"
  RGBcolour Begin
    Red = 65535
    Green = 0
    Blue = 0
  RGBcolour End
Stream End

```

The `Type`, `DatasetDirectory` and `Dataset` entries are the same as for a raster stream. In addition to these, Dynamic Link streams have the following entries.

- **MenuButtonDescription:** The text displayed on the menu button.

- **FilterProgramName:** The name of the translation program to run to generate the PostScript output. Usually these names begin with "ermps\_".
- **LinkType:** The type of PostScript. This can be either "Monocolour" or "Truecolour".
- **FileExtent:** This field determines where ER Mapper will look for the files to be linked to. It may contain a file extension, fixed parameter or program name. This field can be omitted if ER Mapper doesn't need any information about where to find the link.
- **InitialisationProgram:** The program which passes information about the coordinate space and extents of data to and from the link. If this field is absent, ER Mapper assumes that data coming from the link is valid in any coordinate space.
- **CanEdit:** Indicates whether the data in the link can be edited. The entry can be set "Yes" or "No". If the entry is not present it defaults to "No". Currently, the only link which can be edited is the "Annotation overlay".




---

*Information about the Dynamic Link programming interface can be found in ["Dynamic links program interface"](#) in this manual.*

- **RGBcolour Block:** This block defines the color of the vector overlay specified within a dynamic link or classification stream information block. Within the RGBcolour block, there are values for each of the Red, Green and Blue display colors. The example below illustrates a typical entry.




---

*The English spelling of 'colour' must be used in the code.)*

```

RGBcolour Begin
  Red = 65535
  Green = 0
  Blue = 0
RGBcolour End

```

# Filter files (.ker)

Algorithm files can include filter information as described in the [“Algorithm files \(.alg\)”](#) chapter. This filter information is also stored separately as an ASCII (plain text) file. The filter file format is described below.

Standard filters (convolutions), are stored in the subdirectories within the 'kernel' directory. The ASCII filter file has the extension '.ker'. Following is an example of a very simple filter, a 3 by 3 averaging filter.

```
Kernel Begin
  Name= "avg3"
  Description= "3x3 Average filter"
  Type = Convolution
  Rows = 3
  Columns= 3
  OkOnSubsampledData = Yes
  Array = {
    1 1 1
    1 1 1
    1 1 1
  }
  Scalefactor = 9
Kernel End
```

The number of columns in a filter may be different from the number of rows. However, they must both be odd numbers so that every cell is affected by the same number of neighbours on either side, and above and below during processing. Each element of the array may be an integer or a floating point number.

For example,

1x3, 3x1, 3x3, 3x5, 51x37 are valid dimensions

2x2, 1x2, 4x5, 7x10 are not valid dimensions for filters.

Supplied filters are listed in the [“Supplied filters”](#) chapter of the *ER Mapper Applications* manual. User defined filters may be entered using an ascii editor, and stored in one of the subdirectories within the 'kernel' directory with a '.ker' extension.

The entries within the example filter file are described below.

- **Name:** The name of the filter is contained within double quotes. For example,

```
Name = "avg3"
```

- **Description:** The Description entry is displayed in the **Filter** window. For example,

```
Description = "3x3 Average filter"
```

- **Type:** Allowable values for Type are:
  - CONVOLUTION - Normal convolution.
  - THRESHOLD - Convolution with threshold. A value is calculated as per convolution. If the distance between the original value and the new value is less than the threshold, then the calculated value is used, otherwise the original value is used.
  - USER - For filters that incorporate C Code. See the next section.

For example,

```
Type = Convolution
```

- **Rows:** The number of rows in the array. The number of rows must be odd. For example,

```
Rows = 3
```

- **Columns:** The number of columns in the array. The number of columns must be odd.

```
Columns = 3
```

- **OkOnSubsampledData:** This field determines whether a filter is applied to the image before or after subsampling. If an overlay contains a filter with this flag set to NO, then all processing in the overlay will be done before subsampling.

If `OkOnSubsampledData = No` then the filter is applied before subsampling takes place.

If `OkOnSubsampledData = Yes` then the image is subsampled before the filter is applied. This is the default.

- **Array:** The table of values required to filter the data. For example,

```
Array = {
  1 1 1
  1 1 1
  1 1 1
}
```

Each element in the array may be an integer or a floating point number.

- **Scalefactor:** Scale factor for the array. Scale factor may be an integer or floating point number.

```
Scalefactor = 9
```

## C Filters

Filters of type "User" allow ER Mapper to incorporate code provided by the user. An example is shown below. Type "User" filters have the same Description, Rows and Columns fields as described above. The other fields are described below.

```
Kernel Begin
  Description = "Average"
  Type = User
  Rows = 3
  Columns = 3
  UserCodeFileName = "average"
  UserFunctionName = "average"
  InterfaceType = C
Kernel End
```

- **UserCodeFileName:** The file name of a C object file or DLL containing the compiled user code (see next section). For example,

```
UserCodeFileName = "average"
```

refer to the file

```
$ERMAPPER\usercode\filter\c\average.c
```

- **UserFunctionName:** The object file may contain more than one function. The User Function Name is the name of the specific function within the file. For example,

```
UserFunctionName = "average"
```

- **InterfaceType:** The language in which the code is written. Allowable value is:

- C

For example,

```
InterfaceType = C
```



---

See the ["C filters and functions"](#) chapter for more information about C Filters.



# Formula files (.frm)

Formulae are stored in the subdirectories within the 'formula' directory. The ASCII formula file has the extension '.frm'. Following is an example of a very simple difference formula.

```
Formula Begin
  Name = "difference"
  Description = "I1 - I2"
  Formula = "I1 - I2"
  FormulaArg Begin
    StreamInput = 1
    BandNumber = 4
    BandId Begin
      Value = "10.8"
      Width = 1
      Units = "um"
    BandId End
  FormulaArg End
  FormulaArg Begin
    StreamInput = 2
    BandNumber = 1
    BandId Begin
      Value = "0.63"
      Width = 0.1
      Units = "um"
    BandId End
  FormulaArg End
Formula End
```

- **Name:** Name of the formula, as shown in the **Formula** dialog box. For example,

```
Name = "difference"
```

- **Description:** Description entry for the formula.
- **Formula:** The actual formula entry.



*Syntax for the formula is described in the [Editing raster layers](#) and [Formula syntax](#) chapters in the ER Mapper User Guide.*

## FormulaArg Blocks

FormulaArg blocks define the band that is associated with each formula stream input.

- **StreamInput:** Usually associated with a band.

## BandID Blocks

- **BandNumber:** The band number associated with the band in the image.

BandID blocks contain information about the individual bands. All fields are optional.

- **Value:** Value is the spectral value of the band.
- **Width:** Width is the spectral width of the band. The width information is used in the image dataset header (**.ers** files) only.
- **Units:** Units is the type of unit in which the value and width are expressed.

Value and Units are joined together (separated by an underscore) to create the band description. The band description is used for two purposes:

- Providing information about the band.
- To enable ER Mapper to find the correct bands to use when applying formula to an image other than the one for which it was originally created. For example, the values for Landsat TM band 1 would be:

```
Value = "0.485"  
Width = 0.07  
Units = "um"
```

The description displayed by ER Mapper is obtained by joining the value and units separated by a "\_": "0.485\_um"

It is important to note that these fields can be used for types of data other than wavelength. In these cases, width and units may be unavailable or inappropriate.

For example, The values for a copper band of a geochemical image might look like:

```
Value = "ppm_Cu"
```

The description would be "ppm\_Cu". In this case units and width as defined for wavelength data do not make sense.

The `value` field is free format and can be used to put in any description desired. For example, if "ppm\_Cu" is too brief, the `value` field could be set to "parts per million Copper".

In the above example, the ascii formula file was generated by the **Save** option on the ER Mapper **Formula** window. The specific formula contains band information for a particular image, which is also saved.



Formulae are created by way of the ER Mapper user interface, by typing the formula into the **Formula** window and saving the formula. Formula can then be applied to other algorithms, after suitable substitution of the correct bands.



# C filters and functions

Sections on the **Formula** and **Filter** buttons in the [Editing raster layers](#) chapter of the *ER Mapper User Guide*, describe the standard filters and functions supplied with ER Mapper. You can also add your own Convolution and Threshold filters and make up complex formulae using the supplied functions.

This chapter explains how to further expand the available filter and function options by including your own C code. This chapter is only relevant if you have knowledge of C programming or have access to a programmer.

## Filters/kernels

Filters are templates moved over the data to do such things as detect and enhance edges and sharpen or smooth images.



---

*There are many filters supplied with ER Mapper, these are described in the [ER Mapper Applications](#) manual.*

The filters supplied include averaging filters, geophysical filters, sun angle filters, Laplacian, Sobel, sharpening and edge enhancement filters.



---

*You can create your own Convolution and Threshold filter in a plain text file, following the format of filter files detailed in the ["Filter files \(.ker\)"](#) chapter.*

However, you can also create totally new types of filters by including your own C code.

## Writing and compiling usercode on PCs

### Limitations

This section discusses how to write and compile usercode on PC platforms.

- FORTRAN Usercode is NOT supported
- Only Microsoft Visual Studio 2005 is supported

### In the following discussion we assume:

- you have already installed ER Mapper
- ER Mapper is installed in the 'C:\Program Files\ERDAS\ERDAS ER Mapper 7.2' directory.

## Setting and referring to Windows environment variables

To set an environment variable:

```
set ERMAPPER=C:\Program Files\ERDAS\ERDAS ER Mapper 7.2
```

To refer to an environment variable

```
%ERMAPPER%
```

In this section we refer to the directory in which ER Mapper is installed as:

```
%ERMAPPER%.
```

## Checklist

You need the following files supplied with ER Mapper:

- %ERMBIN%\erm\_compile\_usercode.exe — build usercode DLLs
- %ERMBIN%\erm\_compile\_so.exe — used only for dynamic compilation
- %ERMBIN%\make\_def\_file.exe — build .def file from .o files
- %ERMBIN%\erm\_grep.exe — this is GNU grep for Win32
- %ERMAPPER%\lib\win32\dllentry.o — dll entry point object (MSVC)
- %ERMAPPER%\lib\win32\source\dllentry.c — dll entry point source
- %ERMAPPER%\lib\win32\ermapper.lib — main ER Mapper library



---

*%ERMBIN% is %ERMAPPER%\bin\win32*

### Files supplied with Win32 development kit (not from ER Mapper):

- dumpbin.exe — from WIN32 SDK or Microsoft Visual Studio 2005
- nmake.exe, cl.exe and link.exe and associated libraries — complete MSVC development kit

Make files and Windows batch scripts (.bat):

- %ERMAPPER%\bin\pcscripts\erm\_menu\erm\_makeall.bat — driver for usercodebuilds
- %ERMAPPER%\usercode\MakeDll.win32 — build a DLL from c source
- %ERMAPPER%\usercode\kernel\c\Makefile.win32 — build C kernel dlls
- %ERMAPPER%\usercode\formula\Makefile.win32 — build C formula dlls

## Example code

Example source code for the sections explained below can be found in the

`%ERMAPPER%\usercode\formula' and  
`%ERMAPPER%\usercode\kernel\c'

directories. These contain sources that are shipped with ER Mapper and can be used as a starting point when writing your own formulae or kernels/filters.

## Building new formula and filters manually

ER Mapper places its usercode formula files (sources) in the `%ERMAPPER%\usercode\formula' directory. If you want to add any new formula files, these should be placed in this same directory.

The formulas are written as C source (.c) files. These have to be converted to Dynamic Link Libraries (.dll) before ER Mapper can load them.

You can use any other .c file in this directory as an example of how to write a formula. Once you have written a formula file, use the ER Mapper **Utilities / File Maintenance / Filters / Recompile changed C filters** menu option to build the DLL files. Below is the manual procedure for situations where the automatic menu option does not work and for the user who wishes to gain a better understanding of the procedure.

## To edit formula and kernels/filters C source code

From the **Utilities** menu, select either **File Maintenance / Filters/Kernels / Edit a C Formula source file** or **File Maintenance / Formula / Edit a C Filter source file** depending on whether on whether you are editing a formula or filter/kernel.

Select a file to edit from the %ERMAPPER%\usercode\formula or %ERMAPPER%\usercode\kernel\c directory.

When you have completed the editing either save the file or use Save as. to save it under a new name. Make sure that you save it with a .c extension.

## To write or build your own formula manually

1. Open a DOS Prompt window.
2. Set the ERMAPPER environment variable to the directory in which ER Mapper is installed.  

```
set ERMAPPER=C:\Program Files\ERDAS\ERDAS ER Mapper 7.2
```
3. Modify the PATH environment variable to include the ER Mapper PATH.  

```
set PATH=%PATH%;%ERMAPPER%\bin\win32
```
4. Change to the formula source directory.  

```
cd %ERMAPPER%\usercode\formula
```

At this stage you should have a formula file that you want to compile in this directory. Assume the name of this file is form.c
5. Run the nmake command (from Microsoft Visual Studio 2005) to build the DLL.

```
nmake -f %ERMAPPER%\usercode\makedll.win32 DLLSRC=form
```

You should see an output similar to what is shown below:

```
Microsoft (R) Program Maintenance Utility Version 1.50 Copyright (c) Microsoft Corp
1988-94. All rights reserved.

Compiling form.c

cl -nologo -D_X86 -I"c:\Program Files\ERDAS\ERDAS ER Mapper 7.2\include"
-DWIN32

-Dwin32 /W3 /GX /Od -c form.c -Foform.o

form.c

Making DEF file

make_def_file form

Creating library form.lib and object form.exp...
```

At this stage a file named form.dll should be placed in the  
'%ERMAPPER%\usercode\formula\win32' directory.

You can also achieve the same result by running the  
'erm\_compile\_usercode' command. See ["To compile usercode  
directories \(formulae, filters/kernels\)"](#) below.

### **To write and build your own filter/kernel manually**

The previous procedure applies to filters and kernels as well. The  
only difference is the directory in which the source files are located.

- The C sources for the filters/kernels are in  
'%ERMAPPER%\usercode\kernel\c'
- The resulting DLL files are placed in  
'%ERMAPPER%\usercode\kernel\c\win32'

### **To compile usercode directories (formulae, filters/kernels)**

While this procedure is normally run using the menu options, it may  
be necessary for test purposes to compile the usercode sources  
manually. To do this:

1. Open a DOS Prompt window
2. Set the ERMAPPER environment variable by typing:  

```
set ERMAPPER=c:\Program Files\ERDAS\ERDAS ER Mapper 7.2
```
3. Modify the PATH to include ER Mapper executables by typing:  

```
set PATH=%PATH%;%ERMAPPER%\bin\win32
```
4. Change to the usercode directory by typing:  

```
cd %ERMAPPER%\usercode\filters
erm_compile_usercode %ERMAPPER%\usercode\filters
```

## Notes

There are several menu options that manipulate usercode sources. Here we are interested in options that build DLL files from source file. The menu options that perform this task are:

- Utilities/ File Maint/Formula/ Recompile changed User Formulae
- Utilities/ File Maint/Filters/Kernels/ Recompile changed C Filters

These commands compile any new or changed sources in the specified directories.

These three menu options operate on the following three directories that contain ER Mapper usercode respectively:

- %ERMAPPER%\usercode\formula  
contains C formulae
- %ERMAPPER%\usercode\kernel\c  
contains C filters/kernels

Choosing any of these options runs the ER Mapper batch script 'MakeAllUC' (%ERMAPPER%\batch\erm\_menu\MakeAllUC.erb) passing 'formula', 'kernel\c' as a parameter.

This batch script runs

'ERMAPPER\bin\pcscripts\erm\_menu\erm\_makeall.bat' with the full path name of the source directory as a parameter. The following command shows the parameter passed in for compiling formulae sources.

```
erm_makeall.bat %ERMAPPER%\usercode\filters
```

The 'erm\_makeall.bat' file runs 'nmake' which is the make command for Microsoft Visual Studio 2005. The exact command executed is:

```
nmake -nologo -f  
%ERMAPPER%\usercode\filters\Makefile.win32  
\CODEDIR=%ERMAPPER%\usercode\filters
```

There is a Makefile.win32 in each of the usercode directories. These makefiles run the 'erm\_compile\_usercode' command to build the DLLs from the sources. 'erm\_compile\_usercode' is an executable file (.exe) located in \$ERMBIN.

### To manually compile usercode

It may be necessary for test purposes to sometimes compile the usercode sources manually. To do this:

1. Start a DOS Prompt window.
2. Set the ERMAPPER environment variable by typing:  

```
set ERMAPPER=set ERMAPPER=C:\Program Files\ERDAS\ERDAS  
ER Mapper 7.2
```
3. Modify the PATH to include ER Mapper executables by typing:  

```
set PATH=%PATH%;%ERMAPPER%\bin\win32
```

#### 4. Change to the usercode directory by typing:

```
cd %ERMAPPER%\usercode\filters
erm_compile_usercode %ERMAPPER%\usercode\filters
erm_compile_usercode.exe
```

#### Understanding error messages

- Error Category: UserCode / Dynamic Compilation (DC)
- Error Messages (make\_def\_file.c):
- Cannot find dumpbin.exe in PATH
- Cannot find erm\_grep.exe in PATH
- Cannot execute: dumpbin /SYMBOLS <ofile .o|erm\_gnu "SECT"  
.....
- Could not open exports file: <filename
- Could not create DLL EXPORTS file

#### Notes on compiling libermapper programs on PC

Using the ER Mapper C library under Microsoft Visual Studio 2005.  
The following are instructions on how to set up a MSVC project to compile and link against the ER Mapper C library.

They assume:

- Full installation of ER Mapper 7.2 in c:\Program Files\ERDAS\ERDAS ER Mapper 7.2 (substitute the correct directory where necessary).
  - MSVC 4.2 or above is installed on system
1. Start Microsoft Developer Studio.
  2. From the **File** menu select **New**.
  3. Select **Project Workspace** and click **OK**
  4. Enter a project name, for example 'compile\_test'.
  5. Select a project directory, for example 'c:\projects'.
  6. From the **Insert** menu select **Insert Files Info**.
  7. From **Insert Files into project** dialog select: 'c:\Program Files\ERDAS\ERDAS ER Mapper7.2\examples\ER\_MapperLib\_Examples\example1.c
  8. Click **Close** on project files dialog.
  9. Select **Tools** from the menu and click **Options** then select **Directories**.



10. From **Show Directories** select **Directories** and add 'c:\Program Files\ERDAS\ERDAS ER Mapper 7.2\win32' to the directory list.
11. From **Show Directories for list** select **Include Files** and **Add** 'c:\Program Files\ERDAS\ERDAS ER Mapper 7.2\include'.
12. From **Build** select **Settings...** the **Project Settings** dialog will be displayed.
  - Select **C/C++** tab and add **win32** to **Preprocessor Definitions**.



---

*Each item in this list should be comma delimited.*

13. Select the **Link** tab from the **Category** list select **Link** and then **General**.
14. From the **Category** list select **General**.  
Add the following library names to the **Object/Library Modules** field:  
`'ermapper.lib' 'wsock32.lib'`
15. From the **Category** list select **Input**.
16. To the **Ignore Libraries** field add:  
`libc.lib`
17. Click **OK**.
18. From the **Project** menu select **Build compile\_test.exe**.  
The Project will now compile and link. To run the example, go into the directory from a dos shell (the shipped examples require command line options).

## Filter file format

This section describes the filter file formats to use your own C code. User defined filter files should be stored in a subdirectory of the 'kernel' directory. Samples can also be found in the 'kernel' directory. The box below shows the format for a sample file. The syntax for the Description, Rows and Columns is the same as for a standard filter file as set out in the "[Filter files \(.ker\)](#)" chapter. The other fields are described below.

```
Kernel Begin
  Description= "Average"
  Type = User
  Rows = 3
  Columns= 3
  UserCodeFileName = "average"
  UserFunctionName = "average"
  InterfaceType = C
Kernel End
```

Example .ker file for a user defined filter

- **Type:** Must be User.

```
Type =User
```

- **User Code File Name:** This is the file specification of an object file containing the compiled user code (see next section). The path is relative to the 'usercode\filter' directory. For example,

```
UserCodeFileName = "average"
```

refers to the file:

```
ERMAPPER\usercode\kernel\c\win32\average.dll
```

- **User Function Name:** The object file may contain more than one function. The User Function Name is the name of the specific function within the file. For example,
- UserFunctionName = "average"
- **Interface Type:** The language in which the user code is written. Allowable values are:
  - C

For example,

```
InterfaceType = C
```

## The filter Object file

The object (or .dll) file that is executed by the C filter is stored in a subdirectory of the 'usercode\kernel' directory. Sample object files and their corresponding C source files have been stored in 'c' subdirectories as examples, as well as the 'makefile' used to compile them. The file must contain a double precision function with the name specified by the UserFunctionName in the Filter file (see previous section). ER Mapper will pass two integers and one double precision array into the function and return one double precision value. Examples of C function source code are shown below.

```
double average (nr_rows, nr_cols, array)
int nr_rows;
int nr_cols;
double **array;
{
    int r;
    int c;
    double total;
    double nr_elements = nr_rows*nr_cols;
    total = 0;
    for (r = 0; r < nr_rows; r++) {
        for (c = 0; c < nr_cols; c++) {
            total += array[r][c];
        }
    }

    return (total / nr_elements);
}
```

*Example C Function Source Code (average.c)*



*In code written in C values in the array must not be changed within the function. Such changes will produce unpredictable results. Also, the array elements are not stored contiguously in memory but instead take the form of an array of pointers to other arrays.*

## Formula functions

In much the same way that users can incorporate their own code into filters, they can also define functions for processing data within a formula. When ER Mapper encounters a function name in a formula, other than those provided within ER Mapper (and described in the section on the **Formula** button "[Formulae](#)" in the *ER Mapper User Guide*), it assumes it is a C function within an object file of the same name.

For example, the formula:

```
"sum2 (I1, I2) / 2"
```

makes use of the function 'sum2', which is assumed to be in a file called 'sum2.o'. Unidentified functions are reported when the image is displayed. The object code for these functions is stored in the 'usercode\formula' directory.

The user-defined function must return a double precision value. During execution ER Mapper passes the specified number of double precision arguments to the function and returns one double precision number.

### **C user defined function name**

Function names indicate whether the function uses a C interface and can specify the number of arguments to be passed to the function. Thus, the full syntax for a function name is:

```
name [n]
```

where

[n] is the number of arguments

### **Number of Arguments**

To check for errors in the number of arguments passed to the function, include that number in the name of the function. If a different number of arguments is specified in the function call, an error message will result.

### **Type of Arguments**

Function arguments must be declared as double.

### **Examples**

Here is an example function name:

```
"sum(I1, I2, I3)/3"
```

is a name of a function that does not check for the number of arguments.

```
"sum2(I1, I2)/2"
```

calls a C function that requires two arguments to be passed to it.

## **C function format**

Sample C source files and their corresponding object (or .dll) files can be found in the 'usercode\formula' directory as well as the 'makefile' file which was used to compile them.

An examples of source code for functions is shown below.

```
double sum2(a, b)
double a;
double b;
{
return (a + b);
}
```

*Example C Source Code (sum2.c)*

## Initialization and finalization

The C Code options make provision for initialization and finalization routines to be incorporated in the function code. These are each executed once only, before and after the main function is applied to the dataset respectively. Such routines may be used to allocate and deallocate memory or to open and close files.

### Filters

Before calling a function, ER Mapper scans the C object file for initialization and finalization routines. In the filter example above, where `UserFunctionName = "average"` is specified, ER Mapper searches for `'average_init'` and `'average_final'` routines.

If found, the initialization routine will be called first and passed the number of rows, `'nr_rows'`, and number of columns, `'nr_cols'`. Other parameters are available and are described later in this section. A simple example,

```
void average_init (nr_rows, nr_cols)
int nr_rows;
int nr_cols;
{ ... initialization routine code ...
return;
}
```

*Example C initialization routine source code*

Similarly, if it exists, the finalization routine will be called after the function has been executed. No arguments are passed to the finalization routine. For example,

```
void average_final ()
{ ... finalization routine code
...
return;
}
```

*Example C final routine source code*

For C initialization routines there are 3 additional arguments available. These are a text string entered in the **Filter** editor **Filter function name** field (`user_code_params`), a pointer to the current Algorithm structure (`p_a`), and a pointer to the current Stream (or overlay) structure (`p_s`).

This allows User code to extract useful information about the data currently being processed. In particular, this means that user code Filters can get the X and Y cell dimensions for the dataset that is being processed. This is necessary for things like Geophysical filters, and other filters that calculate derivatives such as slopes.

Firstly, when using the dataset cell size, be sure to set the **Process at Dataset Resolution** flag on the Filter, so that you are ensured that the data values that you get in the usercode are non-subsampled data values.



To get the dataset cell size in a C language User code initialization function make sure that you #include "ERS.h" in the User code C source file.

Your user code C filter initialization function looks something like this:

```
#include "ERS.h"

static double global_cell_size_x;
static double global_cell_size_y;

void slope_init(nr_rows, nr_cols, user_code_params, p_a, p_s)
int nr_rows;
int nr_cols;
char *user_code_params;
Algorithm *p_a;
Stream *p_s;
{
    DatasetHeader *p_dsh = p_s->p_cached_dshdr;
    RasterInfo *p_ri = &(p_dsh->u.ri);
    CellInfo *p_ci = &(p_ri->cellinfo);
    double cell_width = p_ci->x;
    double cell_height = p_ci->y;

    /* We now have the cell dimensions - assign them to
    ** local global variables so the processing
    ** function can use at them:
    */
    global_cell_size_x=cell_width; /*Assign to local global */
    global_cell_size_y=cell_height; /*Assign to local global*/
}
```

## Formula

In a similar manner to C filters, before calling a C function, the object code is scanned for an initialization and finalization routine. In this case, the initialization routine is passed the arguments that are passed to the function itself as well as two extra. The two additional arguments are the algorithm pointer and stream pointer described in the section on kernels above.

Thus, a user code Formula function that is declared to have, for example, three parameters, the initialization function is now called with five parameters, the last two being the pointer to the current Algorithm and the pointer to the current Stream. Constants are passed down unchanged, data values are undefined. The finalization routine is called with no arguments.

For example, when the formula 'sum2 (I1,I2)' is specified the following are executed:

```
sum2_init(0,0)
```

```
sum2(I1, I2)
sum2_final()
```

Alternatively, if 'sum2(I1,5)' is specified, the following are executed:

```
sum2_init(0,5)
sum2(I1, 5)
sum2_final()
```

## Linking

### Default switches

The 'config.erm' file specifies default libraries for linking to C code. Thus:

```
CLibrarySwitches = "-lm -lc"
```

The object files may need to be linked with additional libraries. Then the default switches within the config.erm file may be altered to specify the additional libraries to be linked during execution. These switches are the default setting for all user code linking (both filters and formula functions).

For example:

```
"CLibrarySwitches = -L/home/lib -lERS -lm -lc"
```

specifies a user library called libERS.a stored in /home/lib as well as the standard math and C libraries.



---

See "[The ER Mapper configuration file](#)" for detailed information on the **config.erm** file.

### Special case linking

When particular libraries are only used with specific filters or formula functions, you can override the default switch settings by creating a '.link' file of the same name as the function and stored in the same directory as the '.o' file. For example, to execute the function 'test' to be found in the object file 'test.o' which requires links to special libraries, specify the linking switches in the file 'test.link'. The default switches in the 'config.erm' file are overridden. For example, a .link file might contain the following:

```
-L/home/lib -lERS -lc
```

## Special C Function arguments

ER Mapper has two special functions that can only be used as arguments of C functions. These are INPUTVECTOR and STATS. Execution of either of these function arguments requires the dynamic compiler to be turned on, otherwise an error message is displayed.



---

See "[Customizing ER Mapper](#)" in the ER Mapper User Guide for instructions on how to turn the dynamic compiler on and off.



---

*The dynamic compiler is on by default.*

## INPUTVECTOR

- **Syntax:** function name (INPUTVECTOR(input list))
- **Examples:** "USERFUNC (INPUTVECTOR(I1, I2, I3))"

"USERFUNC (INPUTVECTOR(I1..I5))"

The formula INPUTVECTOR passes the following arguments to the user-defined function:

```
int nr_elements;  
double input_vector [];
```

where 'nr\_elements' is the number of items in the input list and the elements of 'INPUT\_vector' are double precision values of the input list.

In the first example above, USERFUNC is called with 'nr\_elements' equal to 3, and 'vector' containing double precision values corresponding to I1, I2 and I3.



## USERSTATS

---

*USERSTATS is supported*

by C interfaces only.

**Syntax: function name (USERSTATS (dataset, region, bandlist))**

**Example:**

```
"userfunc (USERSTATS(ers/Australia_DTM.ers, R1,  
B1..B3))
```



---

*The syntax for the dataset, region and bandlist specifications can be found in the ["Formulae"](#) section of the ER Mapper User Guide.*

The formula USERSTATS passes a pointer to a structure containing statistics for the specified dataset. The structure is defined by the file 'ERSStats.h' which is in the 'include' directory. A user function that is designed to take USERSTATS as an argument must contain the command:

```
#include "ERSStats.h"
```

The 'ERSStats.h' file is shown in the box below. Any of the statistics listed in this structure can be referred to in the user function.



The function `\testustats` is a sample function using `USERSTATS` as an argument. It can be found in the `\usercode\formula` directory.

```

/*****
** Copyright 1990/91 Earth Resource Mapping Pty Ltd. This document contains
** unpublished source code of Earth Resource Mapping Pty Ltd. This notice does not
** indicate any intention to publish the source code contained herein.
** FILE: ERMAPPER/include/ERSStats.h
** CREATED: Tue Feb 12 21:24:31 WST 1991
** AUTHOR: David Hayward
** PURPOSE: The dataset region statistics structure.
*****/
#ifndef ERSSTATS_H
#define ERSSTATS_H
#include "ERStypes.h"
typedef struct ERSStats {
    STRING dsdir; /* dataset header directory */
    STRING dsfname; /* dataset header filename */
    STRING rname; /* region name */
    INT32 nr_band; /* number of bands used to generate stats */
    INT32 *p_bandnr_vec; /* array of band numbers used (note band 1
        ** is 1, not 0)*/
    /*
    ** Following are stored on disk
    */
    INT32 *p_nr_non_null_cells_vec; /* nr of non null cells array */
    INT32 *p_nr_null_cells_vec; /* nr of null cells array */
    double *p_min_vec; /* minimum value array */
    double *p_max_vec; /* maximum value array */
    double *p_mean_vec; /* mean value array */
    double *p_median_vec; /* median value array */
    double **p_p_cov; /* covariance matrix */
    double **p_p_corr; /* correlation matrix */
    double **p_p_inv; /* Inverse of matrix */
    /*
    ** Following are calculated on the fly
    */
    double *p_hectares_vec; /* non-null area in hectares */
    double *p_acres_vec; /* non-null area in acres */
    double *p_cov_eval; /* Covariance eigenvalues */
    double *p_corr_eval; /* Correlation eigenvalues */
    double **p_p_cov_evec; /* Covariance eigenvectors (PCs) */
    double **p_p_corr_evec; /* Correlation eigenvectors */
    double detcov; /* Determinant of covariance matrix */
    double detcorr; /* Determinant of correlation matrix */
    double *p_stdev; /* Standard deviation (using n) */
    double *p_stdevnml; /* Standard deviation (using n-1) */
} ERSStats;

#endif

```

# Creating Color Tables

Color Lookup tables define how ER Mapper will transform digital values to colors in pseudocolor layers.

Color Lookup Table files are held in the "lut" directory. Add any new lookup tables in this directory so that they will appear in the **Color Table** menu in ER Mapper's **Algorithm** dialog box.

The lookup table files are text files, each holding an array of 256 lines of four entries each. The first entry is the pixel lookup value and runs from 0 to 255 down the array. The other three entries are the Red, Green and Blue values that will be used to display that pixel value.

A variety of lookup tables is supplied with ER Mapper. In addition, you can create your own tables tailored to your needs.

## To select a Lookup Table

1. In the **Algorithm** dialog box, select the 'Pseudocolor' **Color Mode**.
2. Select the desired Lookup Table from the **Color Table** drop down list.

## To create a new Lookup Table

1. Create a new color lookup table file.



---

*Use an existing file as a template and follow the structure in the ["Look up table files \(.lut\)"](#) chapter.*

2. To include the lookup table in the lookup table menu, make sure the file is in the "lut" directory and has a '.lut' extension.

## To generate a linear Color Lookup Table

If your color lookup table is to consist of a linear gradation from one color to another, you can use the 'makelut' utility supplied with ER Mapper to help you generate it. 'makelut' creates a Color Lookup Table which adjusts one specified value to another, and outputs the Color Lookup Table to the 'makelut' standard output path. This may be *piped* into an ER Mapper '.lut' file.

To generate a Color Lookup Table:

1. Bring up a Command Prompt window.
2. Type in the CLUT utility command:

```
makelut -n "CLUT_name" iR iG iB nR nG nB >
ERMAPPER\lut\CLUT_filename.lut
```

Where:

- -n "CLUT\_name" is the name of lookup table to be displayed in Lookup Table menu

- `iR iG iB` are the Red Green Blue values of initial color
- `nR nG nB` are the Red Green Blue values of last color
- `ERMAPPER\lut` is the ER Mapper Color Lookup Table directory
- `CLUT_filename` is the file name of color lookup table.

The values are specified in the range 0-255, and the 'makelut' utility correctly generates the values in the 0-65K range.

### Example: a 2-color Lookup Table

For example, to generate a Color Lookup Table that runs from red to blue:

```
makelut -n "Red->Blue" 255 0 0 0 0 255>
ERMAPPER\lut\red_blue.lut
```

Likewise, the command:

```
makelut -n "Brown->Green" 165 42 42 124 255 0 >
ERMAPPER\lut\brown_green.lut
```

generates a brown to lawn green color lookup table which can be useful for vegetation indices.

### Example: a multicolor Lookup Table

The example above is for a two color range. However, the "makelut" program allows for any number of "steps" in each lookup table—the minimum is 2, that is, a start and end point (RGB value). The following example ramps from black to red to green to blue to white in 5 steps.

```
% makelut 0 0 0 255 0 0 0 255 0 0 0 255 255 255 255
```




---

*For information about the Lookup Table file format see the ["Look up table files \(.lut\)"](#) chapter.*

### Example: assigning specific colors to each grey level

There are two ways to do this:

#### **LUT "LoadMethod = Truncate"**

Although there are 256 colors in the LUT table, ER Mapper normally translates those 256 colors into the ones available on your workstation display. For example, on 8-bit displays between 25 and 50 colors are reserved by the windowing system.

ER Mapper normally tries to extract colors over the entire range of colors in the LUT, to result in an image that shows no obvious color holes.

However, for classification type images, this is not what you want. You want the LUT to always use the first N LUT entries, so that LUT color #1 gets mapped to image value 1, LUT color #2 gets mapped to image value 2, and so on.

To do this, make sure that the .lut file you are creating has the following line in it:

LoadMethod = Truncate



See the "[Look up table files \(.lut\)](#)" chapter for more information.

### 1. The "classified image"

Your example image is really a form of "classified" image. Classified images have the following features:

- They have names for each class (data value)
- You can specify a color for each class
- They are simple to display.

The first thing you need to do is ensure that the **.ers** file is set up as a classified image (you could do this as part of the script that creates the **.ers** file).

To look at an example classification image, have a look at:

```
"examples\Shared_Data\ISOCLASS_Landsat_TM_year_1985.ers"
```

The following needs to be put in for each class, in the RasterInfo section:

```
RegionInfo Begin
  Type = Class
  RegionName="SomeDescription"
  RGBColour Begin
    Red = 40000 #note this is 0 to 65535
    Green = 50000 #ditto
    Blue = 20000 # ditto
  RGBColour End
  ClassNumber = 3 # e.g. a number from 1 to 255
RegionInfo End
```



*You don't have to put the color in by editing the file; once the class details are in the **.ers** file, you can update class colors by using the **Edit/Edit Class/Region Color and Name** menu option.*

Once an image has class information (class number and color), to display it you use the new "CLASSIFICATION DISPLAY" layer type. That option takes each pixel value, looks up its color, and displays it in that color.

You can still use formulae within **Classification Display** layers so you can still do "IF I1>3 AND I1<=10 THEN i1 ELSE NULL" to show only classes 4 to 10, or you can do "IF inregion ("some region") and not inregion ("another region") THEN i1 ELSE NULL" to only show the image within some polygon region.

# Look Up Table files (.lut)

The lookup table files describe a mapping of numbers to colors, used to visualize digital data using pseudocolor color mode.

Each of the files contains some information about the color look up table (LUT) and an array of values which describes the red, green and blue entries for each lookup value. The lookup values are in the range 0 to 255 so the entries are in a two dimensional array; 256 lines, each containing:

Lookup value; Red value; Green value; Blue value

The display color values are integers in the range 0 to 65535 which provides 16 bit color entries.

An example '.lut' file is as follows:

```
LookUpTable Begin# lookup table definition block
Version = "5.0"      # *.lut file version
Name = "pseudocolour" # name which appears in
                    # pull-down CLUT selection list
Description = "Standard Pseudocolour"
                    # internal description
NrEntries = 256     # number of entries (lines)in array
LUT = {            # start of LUT array
0      0      0      65535 # first entry (value 0)
1      0      767    65535 # second entry (value 1)
2      0      1535   65535 # and so on ....
.      .      .      .
.      .      .      .
# entries are:
#pixel  red      green      blue values
.      .      .      .
.      .      .      .
.      .      .      .
253    65535    1792      0 # etc
254    65535    1024      0 # until ..
255    65535    256      0 # 256 entries are described
}
                    # close curly braces to
                    # indicate end of array

LookUpTable End
```

## LUT file entries

The Lookup Table file entries are as follows.

- **Version (optional):** The version of the lookup table file. Used internally only. The entry is generated by ER Mapper. For example,

```
Version = "5.0"
```

- **Name (optional):** The entry appearing on the lookup table menu list. If not specified the entry will be set to the file name. For example,

```
Name = "pseudocolour"
```

- **Description (optional):** A short description of the lookup table. This entry is optional. It is not used by ER Mapper. For example,

```
Description = "Standard Pseudocolour"
```

- **LoadMethod (optional):** Obsolete.

- **NrEntries:** The number of entries in the lookup table. This entry must precede the array. The number specified must match the number of lines in the LUT array and is usually 256. For example,

```
NrEntries = 256
```

- **LUT:** The RGB color specification for each of the colors.



---

*For more information about creating color lookup tables, see the ["Creating color tables"](#) chapter.*



# Hardcopy files (.hc)

Hardcopy definition files end with the suffix '.hc' and are found in the 'hardcopy' directory. They are used to specify the attributes of each hardcopy output device.

Hardcopy devices are usually printers, but they can be any type of raster output device such as filmwriters, plotters, photocopiers and standard graphics formats.

The example below illustrates the information required by the hardcopy engine:

```
# HP Deskjet hardcopy device
# This is a 300 DPI black and white ink-jet printer
#
HardCopyInfo Begin # hard copy device info
  Name = "HP DeskJet" # name for device
  DotsPerInchAcross = 300 # horizontal resolution
  DotsPerInchDown = 300 # vertical resolution
  MaximumDotsAcross = 2320 # printable width of paper
  MaximumDotsDown = 3100 # printable height of paper
  FilterProgram = "hetodeskjet"# dithers to get gray scale
  OutputProgram = "lpr -Pdeskjet"# gets data to printer
  Gamma Begin
    Red = 1
    Green = 1
    Blue = 1
  Gamma End
  BackgroundColour Begin# begin background block
    Red = 65535
    Green = 65535
    Blue = 65535
  BackgroundColour End# end background block
HardCopyInfo End # close block
```

Explanations of the entry lines are listed below:

- **Name:** The descriptive name of the hardcopy device, for example:

```
Name = "HP DeskJet"
```

- **DotsPerInchAcross:** The widthwise pixel density in number of dots(pixels) per inch, for example:

```
DotsPerInchAcross = 300
```

- **DotsPerInchDown:** The heightwise pixel density in number of dots (lines) per inch.

```
DotsPerInchDown = 300
```

- **MaximumDotsAcross:** The width of a “page” in pixels. For example,

```
MaximumDotsAcross = 2320
```

- **MaximumDotsDown:** The height of a “page” in pixels. For example,

```
MaximumDotsDown = 3100
```

- **FilterProgram:** The program which converts the raw ER Mapper `ermhe` output into a format suitable for the hardcopy device. If no conversion is necessary, specify “`cat`”. For example,

```
FilterProgram = “hetodeskjet”
```

- **OutputProgram:** The program which actually sends the data to the hardcopy device. For example,

```
OutputProgram = “lpr -Pdeskjet”(Unix only)
```

- **Gamma Block:** This block defines the gamma correction settings for adjusting the red, blue and green components of the output image. This allows you to compensate for variations in colors in printers.

```
Gamma Begin
  Red = 1
  Green = 1
  Blue = 1
Gamma End
```

- **BackgroundColour Block:** This block defines the background color of the output image. There is a value for each of Red, Green and Blue which can be between 0 and 65535. The default background is white which looks like the following.

```
BackgroundColour Begin
  Red = 65535
  Green = 65535
  Blue = 65535
BackgroundColour End
```

- **HardcopyInfo:** Hardcopy definition files use the above hardcopy device attributes contained within the `HardcopyInfo Begin` and `HardcopyInfo End` statements:

```
HardcopyInfo Begin
```

```
HardcopyInfo End
```

- **Comments:** Comments can also be included in the hardcopy definition file, by including a hash (#) symbol as the first character on the comment line.

```
# HP DeskJet hardcopy device  
# This is a 300 DPI black & white ink-jet  
# printer
```

Comments may also be made at the end of lines.

- **Paper Types:** Multiple hardcopy definition files for the one hardcopy device will need to be defined for the different sizes of sheet paper, including continuous paper, used with the hardcopy device. ER Mapper uses a different hardcopy definition file ``.hc`` for each paper size. Change the `MaximumDotsDown` and `MaximumDotsAcross` for the particular paper size.



---

*Additional information about the ``.hc`` files and filter programs can be found in the ["Hardcopy processing and filter programs"](#) chapter in this Manual. A list of devices currently supported by ER Mapper is given in Appendix A "Supported hardcopy formats and devices" in the ER Mapper User Guide .*



# Menu and toolbar files (.erm) and (.bar)

Each of the ER Mapper menus and toolbars is defined by a configuration file. These files are stored in the 'config' directory. The menu and toolbar file structures are virtually identical except that the toolbar file must specify the icon to be included on the toolbar. There are a fixed number of menu files but you can edit them to add new options. On the other hand you can have any number of toolbar files so you can add complete toolbars if you want, as well as editing the existing ones. Any '.bar' files in the 'config' directory are automatically listed in the ER Mapper toolbar menu in alphabetical order.

The files can be edited using your ASCII editor or the editor included in ER Mapper under the **Utilities / User Menu / Edit a File** and the **Utilities / Toolbars / Edit a Toolbar** option.

Example entries from a toolbar file are:

```
"New" "New Image Window" CALLBACK new_alg_cb
"Open" "Open Algorithm into Image Window" CALLBACK load_alg_cb
"Copy_Window" "Copy Window and Algorithm" BATCH "Copy_Window"
"Save" "Save Algorithm" CALLBACK save_alg_cb
"Save_As" "Save Algorithm As" CALLBACK save_alg_as_cb
```

Example entries from a menu file are:

```
"&Raster Cells to Vector Polygons..." "" "erm_rtov -x"
"&Calculate Statistics" "" "ersstats -x"
"C&lassification" "" MENU
    "&Supervised Classification" "" "erm_svclass -x"
    "&ISOCCLASS Unsupervised Classification" "" "erclassx"
    "&View Scattergrams" "" CALLBACK scatter_cb
    "&Edit Class/Region Color and Name..." "" "erm_classedit"
END "" PIN
```

Menu and toolbar entries are made up of three parameters:

- **1. Menu/button option:** For menu files, the text to be displayed on the menu, typed within double quotes. For example,

```
"Classification"
```

For toolbar files, the name of the button picture (.tif) file. The tiff file must be in the 'icons' directory and must be a 24-bit 16 x 16 pixel tiff file. For example,

```
"Mosaic_Datasets"
```

refers to the 'Mosaic\_Datasets.tif' file.

- **2. Tool tips text:** For toolbar files this text is displayed when a user points to the tool button with the mouse cursor. The text must be enclosed in two sets of quotation marks. For menu files the text is not used but the parameter must be present as a minimum of two double quotes. For example,

```
"Zoom Tool"
""
```

- **3. Program call:** A command to be executed when the option is selected. This can be one of the following forms:

- *programname* and switches - to specify an external program. For example,

```
"erm_rtov -x"
```

- **CALLBACK *functionname*** - for calling internal ER Mapper functions. These are listed below. For example,

```
CALLBACK zoom
```

- **BATCH *scriptname*** - for specifying a batch script to be run. For example,

```
BATCH "Copy_Window"
```

If you specify a batch script you can have up to 5 additional parameters to pass into the batch script.

## Submenus

Options in menu files can be grouped together in submenu blocks. The beginning of a submenu is indicated by a menu declaration entry and the end of a submenu is indicated by a menu end statement. The entries in between will appear on the submenu. For example, in the submenu block:

```
"C&lassification" "" MENU
  "&Supervised Classification" "" "erm_svclass -x"
  "&View Scattergrams" "" CALLBACK scatter_cb
  "&Edit Class/Region Color and Name..." "" "erm_classedit"
END ""
```

the menu declaration entry

```
"C&lassification" "" MENU
```

indicates that the following entries should be placed on a submenu and the menu end statement

```
END ""
```

indicates the end of the submenu options. The double quotes must be included. The optional Unix only word 'PIN' specifies that the menu must be able to be pinned up by the user.



- load\_alg\_cb
- load\_alg\_from\_menu\_cb
- load\_any\_cb
- load\_any\_new\_surface\_cb
- load\_vds\_cb
- new\_alg\_cb
- open\_hardcopy\_cb
- open\_window\_cb
- open\_window\_on\_cdev\_cb
- pagesize\_cb
- polarisation\_signature\_cb
- position\_cb
- preferences\_cb
- profile\_cb
- refresh\_cb
- sar\_image\_simulator\_cb
- save\_alg\_as\_cb
- save\_alg\_as\_ds\_cb
- save\_alg\_as\_vds\_cb
- save\_alg\_cb
- save\_image\_to\_clipboard\_cb
- scatter\_cb
- set\_mode\_pointer\_cb
- set\_mode\_roam\_cb
- set\_mode\_zoom\_rect\_cb
- slant\_to\_ground\_conversion\_cb
- speckle\_noise\_reduction\_cb



- stop\_cb
- texture\_analysis\_cb
- zoom\_buttons\_cb

The following are parameters accepted by some callbacks.

- 1
- 2
- 3
- 4
- 5
- ZOOM\_BUTTON\_ALLDS
- ZOOM\_BUTTON\_ALLRASTD
- ZOOM\_BUTTON\_ALLVECD
- ZOOM\_BUTTON\_CONTENTS
- ZOOM\_BUTTON\_CURRENTDS
- ZOOM\_BUTTON\_GEOLINK\_NONE
- ZOOM\_BUTTON\_GEOLINK\_OVERVIEW\_ROAM
- ZOOM\_BUTTON\_GEOLINK\_OVERVIEW\_ZOOM
- ZOOM\_BUTTON\_GEOLINK\_SCREEN
- ZOOM\_BUTTON\_GEOLINK\_WINDOW
- ZOOM\_BUTTON\_PAGE
- ZOOM\_BUTTON\_PREVIOUS
- ZOOM\_BUTTON\_ZOOM\_TO\_WINDOW
- ZOOM\_BUTTON\_ZOOMIN
- ZOOM\_BUTTON\_ZOOMOUT



# Map Composition files (.Idd)

Map Composition items are included in Annotation/Map Composition overlays. Map\_box and map\_polygon objects use the "Attribute" fields to specify the map item and parameters to be interpreted by the PostScript generator. The PostScript for the individual map items is stored in ".Idd" files which define:

- User Parameters, to be specified by the user (for example, text height);
- procedures to be processed to generate PostScript variables and procedures; and
- PostScript include files which contain the main PostScript executable and library code.

Together these supply the PostScript required to display each Map Composition item.

To create a new Map Composition item you need to create a new ".Idd" file and new PostScript include files. Legend Definition ".Idd" files are stored in the directory "legendrules". PostScript include files are stored in the directory "legendps". You also need to store a map item icon to be used in ER Mapper. The icon should have the same name as the ".Idd" file with the extension ".tif" and be stored in the same "legendrules" directory. The tiff file must be a 24-bit 64 x 64 pixel file. You can create this using ER Mapper to print to the "Graphics/TIFF\_64x64\_24bit.hc" hardcopy device.

Map Composition files specify characteristics of a Map Composition item. There are currently twenty types of Map Composition item available, stored in twenty subdirectories of the "legendrules" directory. The directories are:

- Algorithm
- ClipMask
- Dynamic\_Link
- Geology
- Grid
- Image
- Legend\_Item
- Logo
- Map\_Symbols
- North\_Arrow

- PostScript
- Road\_Maps
- Scale\_Bar
- Sea\_Ice
- Symbols
- Text
- Title
- Title\_Block
- Wells
- ZScale

There is also a set of subdirectories containing the ER Mapper Release 4 Map Composition items. These are included to maintain backward compatibility.

An example of a simple .idd file that draws the ER Mapper logo is shown below.

```
LegendRule Begin
  Description= "ER Mapper Logo"
  MapHints Begin
    FastPreviewHint= No
    PageRelativeHint= Yes
    SquareHint= Yes
    WidthHint= 0.3
    HeightHint= 0.15
    LeftMarginHint= 0.0
    RightMarginHint= 0.0
    TopMarginHint= 0.0
    BottomMarginHint= 0.0
  MapHints End
  UserParameter Begin
    Name = "Backdrop Color"
    Type = Color
    Default= "254,254,254"
  UserParameter End
  Generate Begin
    Type = BoundingBox
  Generate End
  PSLibIncludeFiles Begin
    File = "PSLib/lib_Color.ps"
  PSLibIncludeFiles End
  PSExecIncludeFiles Begin
    File = "Logo/ER_Mapper_Logo.ps"
    File = "Logo/ER_Mapper_Logo.eps"
    File = "PSExec/Reset.ps"
  PSExecIncludeFiles End
LegendRule End
```

The “.idd” files are made up of the following entries.

## LegendRule block

The entire “.idd” file is encapsulated in a LegendRule block, so the first line in the file is `LegendRule Begin` and the last line is `LegendRule End`.

- **Description:** The Description entry contains a string description within double quotes. For example,

```
Description = "ERM Logo"
```

- **MapHints Block:** Specifies how a preview version of the map legend is to be displayed.

- **Generate Block:** Specifies procedures to be executed. See "[Generate block](#)" below.
- **UserParameter Block:** These blocks define parameters that the user sets within ER Mapper's Map Composition **Edit Map Item** window, on loading the item. They are converted to variable definitions by the PostScript generation program. The variable definitions have the name specified by the "name" field, the value defined by the "default" field (or that entered by the user), while the "type" field determines the structure of the variable. All fields are required.
- **PSLibIncludeFiles:** The PostScript library procedure definitions. See "[PSLibInclude block](#)" below.
- **PSExecInclude Files Block:** The PostScript files to be included during execution. See "[PSExecInclude block](#)" below.

## UserParameter Block

- **Name:** A text string enclosed in double quotes to become the name of the PostScript variable. For example,

```
Name = "Units"
```

If the name has spaces embedded, these are stripped when forming the corresponding PostScript variable definition.

- **Type:** The "Type" of the parameter determines how the parameter value is processed. The type is specified without double quotes. The following types are allowable and are discussed on the next page:
  - Algorithm
  - YESNO
  - String
  - Number
  - List
  - Color
  - Font
  - DynamicLink

```
Type = List
```

- **AllowedValues Block:** Used to specify allowable values of the parameters of type "List". The values are listed using "Item" specifiers.

```
AllowedValues Begin
  Item = "Miles"
  Item = "Kilometers"
AllowedValues End
```

- **Default:** The value specified in the variable definition unless overridden by the user.

```
Default = "Kilometers"
```

- **DlinkParam Block:** This block must be included for parameters of type "DynamicLink". See the section on [DynamicLink](#) type parameters below.

## Allowable Parameter types

The allowable types are:

### Algorithm

Brings up an algorithm chooser, and converts the name to a PostScript string variable.

### YES/NO

A true/false or yes/no type of value which takes the form of a tick box in the **Edit Map Item** window. This is converted to a PostScript Boolean variable.

```
Name = "Labels At Top"
Type = YesNo
Default = "YES"
```

converts to:

```
/LabelsAtTop true def
```

### String

Takes the text in double quotes and converts it to a PostScript string by enclosing it in brackets ( ). For example,

```
Name = "Title"
Type = String
Default = "Image"
```

converts to:

```
/Title (Image) def
```

### Number

Expects a single integer or floating point number which ermps\_map converts to a double precision number.

```
Name = "Tick Mark Length"
Type = Number
Default = "5"
```

converts to:

```
/TickMarkLength 5.0 def
```

## List

The "Items" in the "AllowedValues" block appear as items in a chooser for the field in the **Edit Map Item** window. The variable is set to one of these string values.

```
Name = "Units"  
Type = List  
AllowedValues Begin  
  Item = "Miles"  
  Item = "Kilometers"  
AllowedValues End  
Default = "Kilometers"
```

converts to:

```
/Units (Kilometers) def
```

Note that the chosen list item is always converted to a PostScript string variable.

## Color

Expects a string value, enclosed in double quotes, consisting of three numbers representing the RED, GREEN and BLUE values for the color, on a scale of 0 to 255. On output, these values are converted to a form suitable for use by the PostScript setrgbcolor operator, being 3 values scaled from 0 to 1. For example,

```
Name = "Color"  
Type = Color  
Default = "128 128 128"
```

becomes:

```
/Color 0.501961 0.501961 0.501961 def
```

## Font

Takes the string that follows and converts it into a PostScript literal string by preceding it with a slash. In the **Edit Map Item** window, this is presented as a menu of the following standard PostScript fonts.

- AvantGarde-Book
- AvantGarde-BookOblique
- AvantGarde-Demi
- AvantGarde-DemiOblique
- Bookman-Demi
- Bookman-DemiItalic
- Bookman-Light
- Bookman-LightItalic
- Courier
- Courier-Bold



- Courier-BoldOblique
- Courier-Oblique
- Helvetica
- Helvetica-Bold
- Helvetica-BoldOblique
- Helvetica-Narrow
- Helvetica-Oblique
- NewCenturySchlbk-Bold
- NewCenturySchlbk-BoldItalic
- NewCenturySchlbk-Italic
- NewCenturySchlbk-Roman
- Palatino-Bold
- Palatino-BoldItalic
- Palatino-Italic
- Palatino-Roman
- Symbol (Symbol)
- Times-Bold
- Times-BoldItalic
- Times-Italic
- Times-Roman
- ZapfChancery-MediumItalic
- ZapfDingbats (ZapfDingbats)

For example,

```
Name = "UnitsFont"
Type = Font
Default = "Times-Bold"
```

becomes:

```
/UnitsFont /Times-Bold def
```

## DynamicLink

Loads the output from a dynamic link into the object bounding box. When this type of parameter is specified a DlinkParam Block must be included specifying the dynamic link chooser and PostScript generation and link initialization programs to run. For example,

```
UserParameter Begin
  Name = "Vector File"
  Type      = DynamicLink
  Default   = "$$, .erv, ermeps_map_ervec, ermeps_ervec"
  DlinkParam Begin
    ChooserString = ".erv"
    LinkProgram   = "ermeps_map_ervec"
    InitProgram   = "ermeps_ervec"
  DlinkParam End
UserParameter End
```

## Generate Block

The Generate Block consists of a list of "Type" fields. Each specifies a procedure to be executed. The procedures use algorithm variables and generate PostScript variables and procedures. Output is in PostScript coordinates (using points or pixels).

Allowable Types are shown below.

## AlgorithmImagePS

Runs an ER Mapper algorithm, and generates a PostScript file that will draw the resulting image. The algorithm that is run must be specified by a user parameter called "AlgorithmName". A PostScript variable ERM\_ALGORITHM\_IMAGE\_PS\_FILE is generated, which contains the name of the PostScript file which contains the algorithm image. This will be drawn into the bounding box of the object by using the PostScript "run" directive. For example:

```
Type = ERM_ALGORITHM_IMAGE_PS_FILE run
```

will draw the algorithm into the bounding box of the map composition item. ER Mapper removes the temporary PostScript file, which is stored in a directory defined by environment variable "\$ERMTMP". For example,

```
Type = AlgorithmImagePS
```

## Progirst

Processed before other map items, typically used for clip masks.

## Linepath

Generates a PostScript procedure "ERM\_LINEPATH".

## Boundingbox

Converts the bounding box coordinates to PostScript point positions and specifies them as four variables:

- ERM\_BBOX\_BBX,
- ERM\_BBOX\_BBY,
- ERM\_BBOX\_BBX and
- ERM\_BBOX\_BBY.

## ScaleBar

Specifies the following variables.

- **ERM\_SCALE:** Floating point value of scale factor, e.g. 1:250000, value of **ERM\_SCALE** would be 250000.000000
- **ERM\_SCALEBAR\_LABELS\_POINTSIZE:** Height (in points = 1/72.27 inch) of labels area above the scale bar proper. This can be used for the point size of the text for the labels, but should be constrained in the actual PSInclude PostScript to be bigger than some minimum, and smaller than some maximum point size.
- **ERM\_SCALEBAR\_UNITS\_POINTSIZE:** Height (in points = 1/72.27 inch) of units area below the scale bar proper. This can be used for the point size of the text for the labels, but should be constrained in the actual PSInclude PostScript to be bigger than some minimum, and smaller than some maximum point size.
- **ERM\_SCALEBAR\_MILES\_CENTER/ERM\_SCALEBAR\_KM\_CENTER(x, y)** position of center of actual scale bar.
- **ERM\_SCALEBAR\_MILES\_HALFHEIGHT/ERM\_SCALEBAR\_KM\_HALFHEIGHT:** Half-height (above and below centerline) of actual scale bar
- **ERM\_SCALEBAR\_MILES\_NR\_DIV/ERM\_SCALEBAR\_KM\_NR\_DIV:** Number of divisions in the scale bar - is almost certainly going to be the same as NumberOfDivisions UserParameter, but **ERM\_SCALEBAR\_KM\_NR\_DIV** and its MILES counterpart should be used in the PostScript.
- **ERM\_SCALEBAR\_MILES\_INFO/ERM\_SCALEBAR\_KM\_INFO:** Array of x-position, y-position of centerline of actual scale bar, positions are of division lines/tick marks.

There is one of these (**ERM\_SCALEBAR\_MILES\_INFO**) for a Miles scale bar, and the other (**ERM\_SCALEBAR\_KM\_INFO**) for a Kilometers scale bar.

- **ERM\_SCALEBAR\_MILES\_LABELS\_INFO/ERM\_SCALEBAR\_KM\_LABELS\_INFO:** Array of: x-position, y-position, text of labels that are to appear above the actual scale bar.



---

*All x and y positions and half-height are expressed in output PostScript units (pixels). The x/y positions for tickmarks are based on the value of the UserParameter "NumberOfDivisions". If this is not set in any UserParameter block, then a default of 3 is used for this.*

### **Special User Parameter "GeodeticScaleLatitude" for scale bars**

The scale bar generation logic will respond to a User Parameter called "GeodeticScaleLatitude". If this User Parameter appears in a legend item definition ".ldd" file, then, for an image in a GEODETIC (Latitude/Longitude) projection, the value of "GeodeticScaleLatitude" will be used to calculate the scale bar parameters. For example, if "GeodeticScaleLatitude" is set to 20, then the scale bar will be generated for latitude 20 degrees (North or South). If there is no "GeodeticScaleLatitude" User Parameter, the scale bar for a GEODETIC projection will be calculated for the latitude of the actual position of the scale bar on the image.

### **ZScale**

- ERM\_ZSCALE\_COLORMAP: Array of (256) colours comprising the colormap, running from lowest to highest index. Note: do not rely on there being 256 colors in the colormap - key any PostScript loops off the length of the array divided by 3. The values for each entry are in the order RED, GREEN, BLUE, scaled to 1.0, suitable for PostScript setrgbcolor operator.
- ERM\_ZSCALE\_LABELS\_INFO: Array of number (between 0.0 and 1.0) and label string. The number is the distance from the bottom of the colormap (scaled to 1.0) of the label, and the label string is the actual label that is to appear at this point on the colormap. There are NumberOfDivisions+1 pairs of entries in this array. NumberOfDivisions is set as a UserParameter - if it is not set, the default is 5.

### **NorthArrow**

- ERM\_NORTHARROW\_ROTATION: A real-valued number of degrees counterclockwise that the northarrow on the output has to be rotated from pointing vertically upward, to correspond to geographic north on the image.

### **Grid**

- GT\_LL\_LABEL;
- LG\_LLLINEGRID;
- LG\_LLTICKGRID;
- GT\_LL\_TICK;
- LG\_ENTEXTGRID;
- GT\_EN\_LABEL;
- LG\_ENLINEGRID;
- LG\_ENTICKGRID;
- GT\_EN\_TICK

## **PSLibInclude Block**

This block lists the PostScript library procedure definitions. These are reusable PostScript routines that are only included once for the whole map (that is, once for all items). It does not include any executable code. The PostScript files are stored in the "legendps" directory.

The library files currently available are:

- libitemlabel.ps - item labelling utility routines
- libstr.ps - character string handling routines
- libtext.ps - text formatting routines
- The convention is to name library include files with a prefix of "lib".

## **PSExecInclude Block**

This block lists the PostScript files to be included during execution. These files contain the actual PostScript executable commands which process the procedures and variables defined in the preceding blocks.

The PostScript files are stored in the "legendps" directory. They may call procedures defined in PSLibInclude PostScript library files.

## Example

In this example the item "Title" is added to an Annotation/Map Composition overlay. The PostScript generation program uses the item definition file "Title.ldd" to help generate the PostScript for the item. The "Title.ldd" file lists the user parameters which are made available to the user as editable attributes. Next, it calls on the PostScript variable generation procedure BOUNDINGBOX to be processed. Then, it includes the PostScript library file "libtext.ps" containing text formatting routines, and the files "setcolor.ps" and "title.ps" which contain executable procedures..

```

LegendRule Begin
Description = "Title"
UserParameter Begin
  Name = "Title"
  Type = String
  Default = "Image"
UserParameter End
UserParameter Begin
  Name = "Color"
  Type = Color
  Default = "128 128 128"
UserParameter End
UserParameter Begin
  Name = "Font"
  Type = Font
  Default = "Helvetica"
UserParameter End
UserParameter Begin
  Name = "Min Font Point Size"
  Type = Number
  Default = "8"
UserParameter End
UserParameter Begin
  Name = "Max Font Point Size"
  Type = Number
  Default = "32"
UserParameter End

Generate Begin
  Type = BOUNDINGBOX
Generate End

PSLibIncludeFiles Begin
  File = "libtext.ps"
PSLibIncludeFiles End

PSExecIncludeFiles Begin
  File = "setcolor.ps"
  File = "title.ps"
PSExecIncludeFiles End
LegendRule End

```

An example PostScript include file `setcolor.ps` contains the following:

```

%!
Color setrgbcolor

```

An example PostScript include file "title.ps" is as follows:

```
%%
%% Place the title in the centre of the bounding box defined
%% with a font size to fill the box or use the minimum or
maximum
%% font size to prevent the text becoming too small or too large
%%

Color setrgbcolor

% set variables for box width and height
/box_ht ERM_BBOX_TRY ERM_BBOX_BLY sub def
/box_wdth ERM_BBOX_TRX ERM_BBOX_BLX sub def

% set the font size to fit the box
Font MinFontSize MaxFontSize box_ht box_wdth Title
FitFont

% display the text in the centre of the box
ERM_BBOX_BLX box_wdth 2 div add
ERM_BBOX_BLY height Title StringHeight sub 2 div add
moveto
Title centertext
```

The following PostScript is generated by ER Mapper to produce the "Title" item.

(many pages of PostScript created by the PostScript generation program setting up common variables and procedures)

**% PostScript variables created from UserParameter specifications in "Title.idd" and % edited object attributes**

```
/Title (Australia DTM) def
/Color { 1.000000 0.117647 0.000000 } def
/Font /Helvetica def
/MinFontSize 8.000000000000000000 def
/MaxFontSize 32.000000000000000000 def
```

**% PostScript variables defined by the Generate BoundingBox command in "Title.idd"**

```
/ERM_BBOX_BLX 40 def
/ERM_BBOX_BLY 39 def
/ERM_BBOX_TRX 234 def
/ERM_BBOX_TRY 93 def
```

...

(a few pages of code included from "text.ps" containing common text variables and procedure definitions)

...

**% PostScript code included from "startup.ps"**



```

%!
Color setrgbcolor

% PostScript code included from "title.ps"
%%
%% Place the title in the centre of the bounding box
defined
%% with a font size to fill the box or use the minimum
or maximum
%% font size to prevent the text becoming too small or
too large
%%
Color setrgbcolor

% set variables for box width and height
/box_ht ERM_BBOX_TRY ERM_BBOX_BLY sub def
/box_wdth ERM_BBOX_TRX ERM_BBOX_BLX sub def

% set the font size to fit the box
Font MinFontSize MaxFontSize box_ht box_wdth
Title FitFont

% display the text in the centre of the box
ERM_BBOX_BLX box_wdth 2 div add
ERM_BBOX_BLY height Title StringHeight sub 2 div add
moveto
Title centertext

% Concluding PostScript generated by "ermeps_map"
GR
% End of Generated PS
GR
showpage

```



# Dynamic Links

Dynamic Links are used to access data from sources outside ER Mapper, such as vector or tabular data from a GIS or a database system, and display it graphically using PostScript. The 'dynamiclinks.erm' file defines the menu of Dynamic Links options available. It is stored in the 'config' directory. Edit this file to add new links. It can be edited using the **File Maintenance / Configuration / Edit Dynamic Links Menu file** option from the **Utilities** menu.



---

See the [Dynamic Links \(vector layers\)](#) chapter in the ER Mapper User Guide for information about using the available Dynamic Link options. In this manual, see the ["Dynamic Links program interface"](#) chapter for creating new Read Only and Read/Write dynamic link interfaces.

## Menu entries

Menu items are specified by single line entries with up to 8 parameters, 5 required and 3 optional. If any optional parameter is specified all previous optional parameters must also be specified (as null parameters "" if necessary) so that the parameter number is correct.

Selected entries from the file are shown below.



*In the examples below some of the entries wrap around because they are wider than the page. However, they must be entered as single lines in the file.*

```
"Contours" "Contours" "ermepscontours" TRUECOLOUR
"$$$SCRIPT=Dlink_Contours.erb $DEFAULT $$MANAGERRELATIVEPATHS"
"ermepscontours"
# [03]
"Grid Datasource Points" "Grid Datasource Points"
"ermeps_display_grid_datasource" MONOCOLOUR
"$$$SCRIPT=DlinkDisplayGridDatasourcePoints.erb"
"ermeps_display_grid_datasource"
"Grid Datasource TIN" "Grid Datasource TIN"
"ermeps_display_grid_datasource" MONOCOLOUR
"$$$SCRIPT=DlinkDisplayGridDatasourceTIN.erb"
"ermeps_display_grid_datasource"
# end [03]
"AutoCAD DXF" "DXF Link" "ermeps_dxf" TRUECOLOUR ".dxf"
"ermeps_dxf"
"MicroStation DGN File" "DGN Link" "ermeps_dgn" TRUECOLOR
".dgn" "ermit_dgn"
"External Vector formats" "" MENU
"GeoImage" "GeoImage Link" "ermeps_geoimage" MONOCOLOUR ".geo"
"ermeps_geoimage"
"Geocoded Monocolor PostScript" "Geocoded Monocolor
PostScript Link" "ermeps_erp" MONOCOLOUR ".erp" "ermit_ers"
"Geocoded Truecolor PostScript" "Geocoded Truecolor
PostScript Link" "ermeps_erp" TRUECOLOUR ".erp" "ermit_ers"
END
"Tabular Data" "" MENU
"Symbols" "Symbols Link" "ermeps_symbols" TRUECOLOUR ".txt"
"Table of data shown as Circles" "Table of Data as Circles
Link" "ermeps_table_circle" MONOCOLOUR ".tbl" "ermit_en0"
"EDIT" "erm_link"
"Test R/W link" "Test link for Geodass query"
"ermeps_table_circle" MONOCOLOUR ".tbl" "ermit_en0" "EDIT"
"0_erm_link"
"Table of Circles with Rotation" "Table of Data with
Rotation Link" "ermeps_table_circle" MONOCOLOUR ".tbl"
"ermit_any" "EDIT" "erm_link"
"Table of data shown as Outline Circles" "Table of Data -
Outline Link" "ermeps_table_outline" MONOCOLOUR ".tbl"
"ermit_en0" "EDIT" "erm_link"
END
```

## Submenus

Menu items can be grouped into submenus. The beginning of a submenu is indicated by a menu declaration entry and the end of a submenu is indicated by a menu end statement. The entries in between will appear on the submenu. Submenu entries are indented by convention, this is not necessary for ER Mapper to read them.

In the example above, the entry:

```
"External Vector formats" "" MENU
```

defines a submenu block. The first parameter "External Vector formats" defines the text to appear on the menu. The second parameter is a pair of double quotes "". The third parameter is the keyword MENU. The lines that follow define the available links, and the submenu is terminated with the END entry.

## Menu entry parameters

The parameters in a menu entry are:

1. **Menu option:** The text that is displayed on the menu button, typed within double quotes. For example,

```
"MicroStation DGN File"
```

2. **Description:** A description field enclosed in double quotes. In the Algorithm window, when you add a dynamic link layer, this is shown as the layer name in the layer list. This parameter is required but can be just two sets of double quotes. Examples are:

```
"DGN Link"  
""
```

3. **PostScript generation program call:** The name of the translation program to extract the data and generate the PostScript. By convention, the names of these programs begin with "ermps\_". For example,

```
"ermps_dgn"
```



---

See "[PostScript generation](#)" above.

4. **PostScript type:** This can be:

- MONOCOLOUR
- TRUECOLOUR

MONOCOLOUR means the resultant PostScript will be shown as a single color. TRUECOLOUR shows full color PostScript but is slower to display.

For example,

```
MONOCOLOUR
```



---

"COLOUR" must have the English spelling.

- 5. Link chooser (optional)** Often some sort of user selection is needed to specify what data is to be extracted from the link. This parameter specifies how the source data is to be chosen. It may contain one of six types of entries. For example,

```
$$CHOOSEER=arc_chooser $DEFAULT"  
".dxf"
```



---

See the "[Link chooser parameter](#)" and "[Chooser program](#)" sections.

- 6. Initialization program (optional)** The name of the program which compares information about the current ER Mapper coordinate space and extents with the same information from the target data source. By convention, the names of these programs begin with "ermit\_". If this field is not specified, ER Mapper assumes that the data coming from the link is valid in any coordinate space.



---

The initialization program is described in the "[Link initialization](#)" section.

- 7. EDIT flag (optional)** This parameter is set to "EDIT" if the data can be edited or annotated within ER Mapper. Syntax is:

```
"EDIT"
```



---

See the "[Dynamic Links program interface](#)" chapter for information on editable links.

- 8. Edit program (optional)** The edit program to run. This parameter must be present if the EDIT flag is included. For example,

```
"erm_link"
```

## To add a Dynamic Link option

1. Decide where on the menu you want the option to appear.
2. Edit the "dynamiclinks.erm" file using your ASCII editor or by selecting the "**File Maintenance / Configuration / Edit Dynamic Links Menu file**" option from the **Utilities** menu.
3. Insert menu entry in the position you want it to appear. Your new item will appear on the **Edit / Add Vector Layer** menu in the **Algorithm** dialog box.

## Link chooser parameter


This field specifies how the source of data for the dynamic link is to be selected. It may contain one of seven types of entries. (The choice made during this step is passed to the initialization and PostScript generation programs via the fifteenth argument. See "[Link initialization](#)" above. The information that gets passed is indicated for each case below.)

ER Mapper uses the link chooser parameter to determine whether to use relative or absolute path handling.

- **No Parameter:** When there is no choice to be made and no parameter needs to be passed to the conversion program this field is totally omitted (or consists of empty quotes ""). The syntax is:

```
""
```

Nothing is copied to Argument 15.

- **Dataset Chooser:** A field beginning with a "." indicates the standard ermapper dataset chooser should be used. The icon  appears in the process diagram for the layer. It displays all the files with the specified extension from the the current image dataset directory and subdirectories. For example,

```
".erv"
```

ER Mapper uses relative path handling for this chooser.

The file specification of the selected file is passed to Argument 15.

- **Fixed Parameter:** When a fixed argument needs to be passed to the conversion program, without the user making any choice, a dollar sign "\$" precedes the text to be passed to the conversion program. For example,

```
"$TODAY"
```

The text after the \$ is passed to Argument 15.


ER Mapper will never use relative path handling for this parameter.


- **\$\$ALG Parameter:** If the keyword \$\$ALG is specified, the file specification of a temporary file containing the current algorithm is passed. When ER Mapper saves the algorithm to a temporary file it converts all path names to absolute.

```
"$$ALG"
```

- **Link Chooser:** The name of a list generating program for choosing other types of files or possible GIS layers, database tables or other selections. The program name must not begin with "." or "\$" and must output its list of choices to stdout. For example,

```
"erm_choose_en_grid"
```

The link chooser icon  is included in the process diagram for the layer and displays the list. The file specification of the selection is passed to Argument 15.

- **External Link Chooser:** For certain links, the Dataset and Link Choosers may not be flexible enough. With this option you can specify an external chooser program. The link chooser icon  is included in the process diagram for the layer and displays the list. The parameter starts with the keyword sequence "\$\$CHOOSER=" and is followed by the name of the chooser program and its arguments. Thus,

```
"$$CHOOSER=chooserprog arg1 arg2"
```

where *chooserprog* is the program to execute and *arg1* and *arg2* are arguments to *chooserprog*.

\$DEFAULT is a special argument keyword. It is replaced with the last value returned by the chooserprog. Thus,

```
"$$CHOOSER=chooserprog arg1 $DEFAULT"
```

\$\$MANAGERELATIVEPATHS is another special argument keyword which indicates that file or directory specifications that the link wants ER Mapper to manage are prefaced by "file=" or "dir=" in the argument string. The arguments are separated by the pipe "|" symbol. When saving an algorithm, ER Mapper scans the argument string for "file=" or "dir=" and converts the subsequent string to a relative path if possible. When re-loading the algorithm, ER Mapper converts the file or directory specification back to an absolute path and passes this back to the link.

To use quotes to specify an argument with imbedded spaces, precede them with backslashes (\). For example,

```
"$$CHOOSER=erm_xgettext -p \"Enter the cover\" -s 30"
```

ER Mapper executes the chooser program when the user presses the link icon. ER Mapper hangs while the chooser program runs. When the chooser program exits, ER Mapper reads its stdout and uses this string as the argument to pass to the link (instead of a file name).

The argument string has no fixed format; it is up to the dynamic link program to interpret it.

ER Mapper will handle backslashes, quotes, and newlines in the argument string, but it must be no longer than 512 bytes. If more information is required, it is suggested that the information be put in a file, and the name of the file passed to the link programs.

The handling of non-printable characters in the argument string is currently undefined.



- **Batch Script Chooser:** With this option you can specify a batch script to run to determine the source of the data. The syntax is:

```
"$$SCRIPT=batchfile.erb"
```

This link can also use the \$DEFAULT and \$\$MANAGERRELATIVEPATH keywords. The syntax is:

```
"$$SCRIPT=batchfile.erb [$DEFAULT]  
[$$MANAGERRELATIVEPATHS]"
```



# Digitizer files

There are three types of digitizer files used by ER Mapper. They are:

- the digitizer type file `.dtp` stored in the Program Files\ERDAS\ERDAS ER Mapper 7.2\digitizer\_type directory
- the digitizer configuration file `.dcf` stored in the Program Files\ERDAS\ERDAS ER Mapper 7.2\digitizer\_config directory
- the digitizer session file `.dig` stored in the Program Files\ERDAS\ERDAS ER Mapper 7.2\digitizer\_config directory.

The syntax of these files is described in this chapter.

## Digitizer type file .dtp

This file contains the format information needed to decode the input stream from the digitizer into numbers that ER Mapper can work with. For example,

```
DigTypeInfo Begin
  Comments= "Format_5: FS+XXXXXX+YYYYY<CR><LF>"
  DigStringLength = 16
  LastCharacter = "NL"
  XPosition= "2,7"
  YPosition= "8,13"
  FlagPosition= "0,0"
  StatusPosition= "1,1"
  ButtonDown = "D"
  ButtonUp = "U"
DigTypeInfo End
```

The entries are as follows.

- **Comments:** A comments field for information. In this example it indicates the string that the digitizer transmits. For example:

```
Comments = "Format_5: FS+XXXXXX+YYYYY<CR><LF>"
```

- **DigStringLength:** The number of characters in a complete string. For example,

```
DigStringLength = 16
```

- **LastCharacter:** The last character of the string. Allowed options are :
  - "CR" (Carriage Return)
  - "LF" (LineFeed)

- "NL" (Newline)

For example,

```
LastCharacter = "NL"
```

- **XPosition:** The position of the x coord substring. For example,

```
XPosition= "2,7"
```

- **YPosition:** The position of the y coord substring. For example,

```
YPosition= "8,13"
```

- **FlagPosition:** the position of the flag substring. For example,

```
FlagPosition= "0,0"
```

- **StatusPosition:** The position of the status substring. For example,

```
StatusPosition= "1,1"
```

- **ButtonDown:** The button down character. For example,

```
ButtonDown = "D"
```

- **ButtonUp:** The button up character.

```
ButtonUp = "U"
```

## Notes

- This particular format is one of the 30 or so formats specified in the Altek manual for one of their tablets. The status substring is usually indicates whether the button was down or up. The Flag Position is generally used to indicate which button was pressed on the digitizer mouse (they can have up to 16 different buttons).
- This information is used as a "set of rules" to decode the digitizer string into a suitable ER Mapper coordinate, e.g. the DigString length field tells us when we have grabbed a digitizer "packet", the last character lets us check the packet to make sure it is not corrupted, and the x and y position fields tell us where to look for the x coord and the y coord, and so on. This gives a great deal of flexibility when dealing with the multitude of different types of tablets.

## Format of .dtp files.

There are 3 example .dtp files:

- ERMAPPER\digitizer\_type\Calcomp.dtp
- ERMAPPER\digitizer\_type\GTCO.dtp
- ERMAPPER\digitizer\_type\Altek.dtp

These are for the following formats. Other formats will work, as long as they conform to the 3 rules:

- ASCII output
- last character <cr> or <lf> or <nl>
- a button down/up character (this can be relaxed to just a button up character) for stream mode.

### Calcomp digitizers

Format #0

Format Name Calcomp 2000-A

Format is XXXXXX,YYYYY,C CR [LF]

The format varies with the tablet size and the resolution. ER Mapper will work with these other formats, but will need a new .dtp file. The default shipped Calcomp digitizer type file is for the following tablet size/resolution combinations.

i.e. For the following combination of tablet size and resolution the format should be the one shipped (i.e. XXXXX,YYYYY,C,CR [LF])

Tablet size	Resolution
A,B	> 508
C	> 401
D,E,J	<= 1279

Size A - 12" x 12"

Size B - 12" x 18"

Size C - 18" x 24"

Size D - 24" x 36"

Size E - 36" x 48"

Size J - 44" x 60"

Consult your digitizer manual, if you want to use another lpi or format.

### GTCO digitizers

PXXXXX<SP>YYYYY<CR><LF>

### Altek digitizers

Format number 5: FS+XXXXX+YYYYY<CR><LF>

Consult your digitizer manual on how to set the digitizer switch settings.

ER Mapper accepts digitizer input for:

- Defining Rectification control points
- Cell Coordinate information
- Cell Profile
- Annotation system: for creating polygons, polylines and points, creating map composition polygons, and creating raster regions.

For the input to show up on the image and/or dialog, the current destination must exist and must have the same coordinate space as the map on the digitizer.

The exception to this is the Rectification Control Point chooser which accepts the digitizer input without a "TO" image.

## Digitizer configuration file .dcf

This file contains the communication parameters such as the port that the digitizer is connected to and the baud rate. This file is normally created using the Digitizer Config dialog from within ER Mapper. An example file is shown below.

```
DigConfigInfo Begin
  Name= "/digitizer_config/Altek.dcf"
  Port  = "/dev/ttya"
  BaudRate= 19200
  Parity= Odd
  StopBits= 2
  DataBits= 7
DigConfigInfo End
```



---

*ER Mapper accepts forward (/) slashes for directory path entries in files regardless of whether it is the PC or Unix version.*

The entries are as follows:

- **Name:** The name of the file on disk.

```
Name = "/erm/ermapper_dev/digitizer_config/Altek.dcf"
```

- **Port:** The serial port to which the digitizer is connected. For example,

```
Port = "/dev/ttya" (Unix)
Port = "com1" (PC)
```

- **BaudRate:** The baud rate of the connection. For example,

```
BaudRate = 19200
```

- **Parity:** The parity of the data, for error detection. Allowable values are Even or Odd or None. For example,

```
Parity = Odd
```

- **StopBits:** The number of bits after the data byte signifying the end of the data byte. Must be 1 or 2. For example,

```
StopBits = 2
```

- **DataBits:** The number of bits in the data byte. Must be 7 or 8. For example,

```
DataBits = 7
```

## Digitizer session file .dig

The digitizer session file contains the information about a particular session. It records information specific to the map being digitized such as the map projection. An example .dig file is:

```
DigSessionInfo Begin
  Name = "/digitizer_config/session1.dig"
  Comments= "This is a test for the diginfo functions"
  TypeFile= "/digitizer_type/Altek.dtp"
  ConfigFile= "/digitizer_config/Altek.dcf"
  WarpControl Begin
    CoordinateSpace Begin
      Datum = "AINABD70"
      Projection= "CECASP"
      CoordinateType= LL
      Units = "METERS"
      Rotation= 0:0:0.0
    CoordinateSpace End
  WarpControl End
DigSessionInfo End
```

The entries in the .dig file are as follows:

- **Name:** The name of the file. For example,

```
Name = "/digitizer_config/session1.dig"
```

- **Comments:** A comments field. For example,

```
Comments = "This is a test for the diginfo functions"
```

- **TypeFile:** The digitizer type file. For example,

```
TypeFile = "/digitizer_type/Altek.dtp"
```

- **ConfigFile:** The digitizer configuration file. For example,

```
ConfigFile = "/digitizer_config/Altek.dcf"
```

- **WarpControl Block:** This contains information about the map projection of the map you are digitizing from. For example,

```

WarpControl Begin
  CoordinateSpace Begin
    Datum = "AINABD70"
    Projection= "CECASP"
    CoordinateType= LL
    Units = "METERS"
    Rotation= 0:0:0.0
  CoordinateSpace End
WarpControl End

```

## Notes

- The session (.dig) and config (.dcf) files are created via the Digitizer Session dialog and the Digitizer Config dialog respectively.




---

See the ["Installing and configuring digitizers"](#) chapter in the *ER Mapper Installation manual*.

- The type (.dtp) file is created by the user. There is no dialog to do this, although this may change at a later date. Some sample files are shipped with ER Mapper, so all that is needed is some simple file editing.

## Digitizer modes

ER Mapper requires the digitizer to transmit coordinates in ASCII format.

ER Mapper supports two digitizing modes: *Point* mode and *Line* mode. Both of these modes are activated by a "button down" type of event.

Some digitizers also have a *Continuous* mode where points are transmitted continuously if the digitizer is on. These modes are NOT supported - they are treated as Point mode and will most likely prevent the mouse from being used (ER Mapper treats digitizer events as being identical to mouse events over an image window).

In order to use Line mode, particular formats must be used. The format MUST include a cursor status character indicating if the cursor (or pen or button) is UP or DOWN. If there is no up or down status character in the format then all digitizer events will be treated as points.

These requirements can be interpreted as "You must have these configuration switches set this way" and should not be a restriction on the range of tablets that are supported.

ER Mapper does not accept digitizer input which uses <CR> as the end of coordinate character.



# Hardcopy Processing and Filter programs

Images are sent to hardcopy devices by a program called 'ermhe'. This program can be run stand-alone, but is usually started from within ER Mapper using the **File** menu **Print...** option. See the [Printing](#) chapter in the *ER Mapper User Guide*. Because the hardcopy utility is a separate program to ER Mapper, you can continue using ER Mapper while an image is being sent to a hardcopy device.

This chapter provides detailed information about the internal formats used by the hardcopy engine and how to generate new hardcopy filters. It is not necessary to read this section in order to print using ER Mapper.

## Hardcopy processing stages

Internally the ER Mapper Hardcopy Engine generates hardcopy output in three stages.

### Create device independent output

The first stage creates output for the device at the resolution of the device. This output is always 24 bit RGB, regardless of the capabilities of the output device. This first stage also merges the vector overlays with the raster overlay components of the algorithm. The result is that a true raster file is represented in absolute Color values. This first stage is device independent.

### Convert to device format

The data from the first stage is piped into the device dependent stage of the hardcopy output. This piping process ensures that huge amounts of disk storage are not required for an intermediate output file.

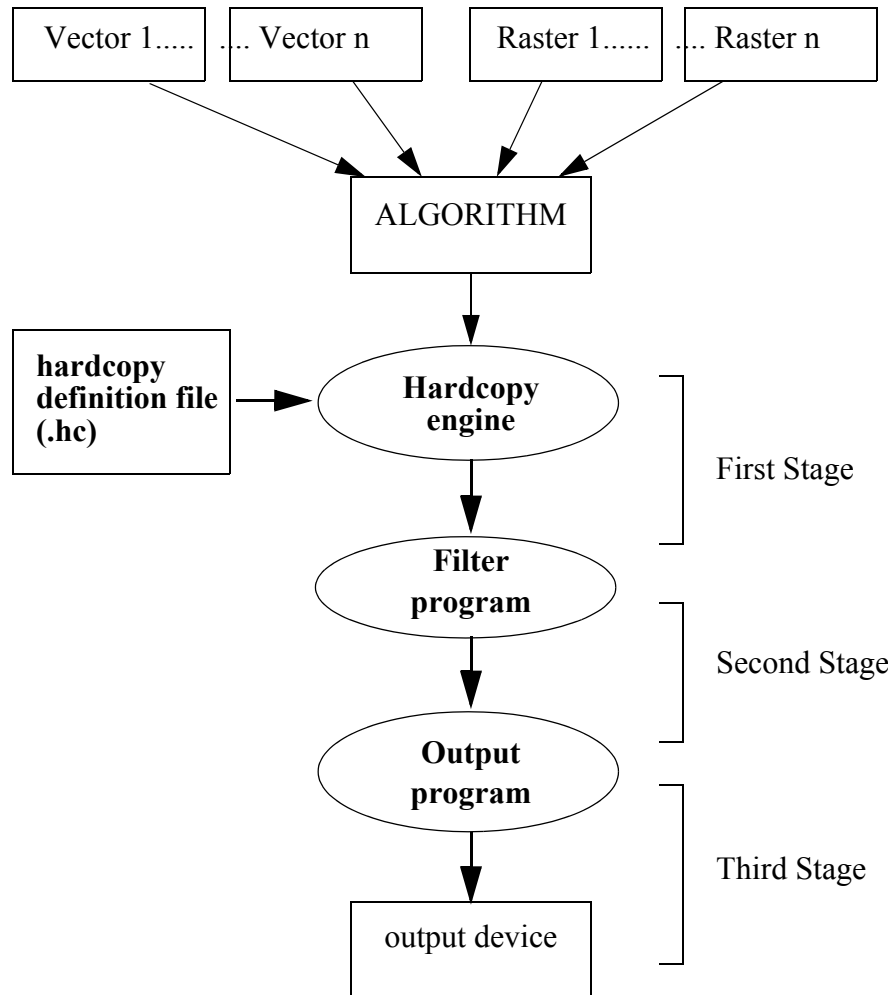
In the device dependent second stage the RGB values per pixel are converted into the hardcopy device specific format.

### Send to device

The third stage involves piping the hardcopy device-specific format to the hardcopy device. Output can be directed to any destination, such as a real output device, or a magnetic tape unit or simply a file on disk.

Output to a hardcopy device involves 3 programs: ermhe, a filter program, and an output program. The filter and output programs used are specified in the hardcopy control files which are stored in the 'hardcopy' directory.

These three stages are shown diagrammatically below.



Each scanline of the algorithm (processed image) is piped through all stages so that the output of one stage becomes the input of the next stage of hardcopy processing. This enables very large images omages with much processing to be output to the hardcopy device without creating huge temporary files.

### Strip printing

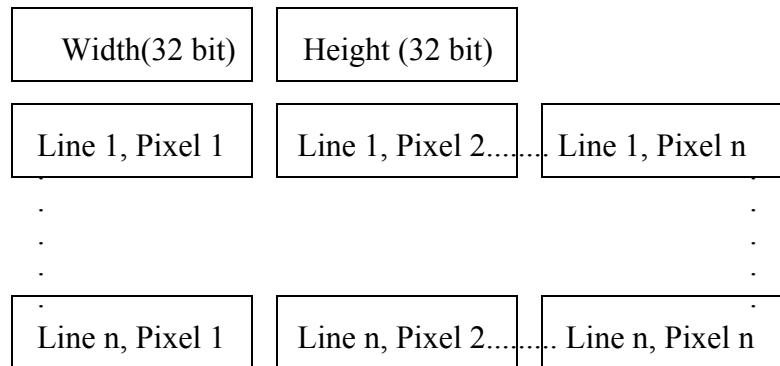
If your hardcopy Mapperpper will automatically divide the output image up into strips or pages that are no larger than the maximum size allowed by the hardcopy device. Each of these strips or pages are then processed individually.

### The hardcopy engine

The hardcopy engine gathers information about a hardcopy device by looking for a file with suffix '.hc' in the ERMAPPER\hardcopy directped into the filter program.

## The hardcopy engine Output Format

The hardcopy engine uses an internal standard for processing of hardcopy imagery, which is then converted to the output type required. This internal format is:



The hardcopy engine outputs one 4 byte integer (UINT32 or long word) for each output pixel. The first two long words specify the width and height of the raster file. The hardcopy engine guarantees the width and height specified for a hardcopy image will never exceed the maximum width and height allowed for the hardcopy device selected. Each long word thereafter represents the hardcopy image data. Line 1, Pixel 1 refers to the top left hand corner of the image.

Each 32-bit value comprising the actual image contains a value from 0 - 255 for red, green and blue respectively, and a byte indicating the height value for algorithms with a height layer. Assuming that byte 4 is the most significant byte, the output integer can be interpreted as follows:

	Byte 4	Byte 3	Byte 2	Byte 1
	Height (if relevant)	Red	Green	Blue
Value	(0-1)	(0-255)	(0-255)	(0-255)

## Filter Program

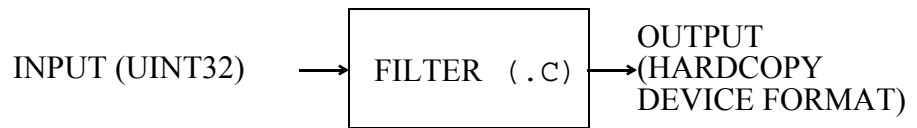
The filter program takes the data in hardcopy engine format and converts it into the format required by the hardcopy device. The output from the filter program is then piped into the output program.

### Creating a Filter Program (.c)

ER Mapper includes a library of many hardcopy device drivers. We support as many output devices as possible and continually add new devices, so please contact your distributor or Earth Resource Mapping if your hardcopy device is not listed.

If you wish to create a Filter Program the interface is documented in this section.

The filter takes as standard input the normal Hardcopy Engine output, which consists of two UINT32 specifying output width and height, followed by multiple UINT32's, one per output pixel.



### Filter program

Please consult your hardcopy device manual for details of the hardcopy device output format your printer requires.

Standard output filters are provided for many popular printers.

Following is an example Filter Program (.c) for the Sharp JX730 color ink jet printer (hetojx730.c).

```
/*
*****
***
** Copyright 1989 Earth Resource Mapping Pty Ltd.
** This document contains unpublished source code of
** Earth Resource Mapping Pty Ltd. This notice does
** not indicate any intention to publish the source
** code contained herein.
**
** FILE:
/home/erm/erm/src/erm/he/hetojx730/hetojx730.c
** CREATED:Mon Jul 23 19:28:40 WST 1990
** AUTHOR: Stuart Nixon
** PURPOSE:Hardcopy To sharp jx-730 filter program
(printer similar to tektronix)
** If the define variable FORM_FEED is true, then a form
feed is used,
** otherwise a 2" gap is printed at the end of a page...
**
** COMMENTS:
**
** 1. We use the Unidirectional mode, which is slow but
produces high
** quality prints.
**
** 2. This filter takes as standard input the normal
Hardcopy
** Engine output, which consists of two UINT32's
specifying
** output width & height, and then multiple UINT32's,
one per output pixel.
*/
```

```

** The UINT32 consists of "xRGB", which "x" is currently
unused.
** This filter converts that output into something the
printer
** can understand.
**
** 3. This filter dithers output into a ordered 4x4
matrix in order to
** get enough colours.
**
** [01] 21/02/89 snsAdded formfeed at end of a strip
** [02] 23/07/90 snsCopied from ColourQuick &
supporting vectors
** [03] 02/08/90 snsMerged the 2" gap and the FORM
FEED versions
** [04] 03/08/90 snsNow doing compressed print mode
where possible
*****
**/

#ifndef lint
static char *sccsident="@(#) %M%:%I% %D% Copyright
Earth Resource Mapping 1989";
#endif

#include <stdio.h>
#include "ERS.h"

/*
** CONSTANTS
*/

#define DITHER4
#define VECTOR_FLAG_SHIFT24/* [04] */
#define RED_SHIFT16
#define GREEN_SHIFT8
#define BLUE_SHIFT0

#define STDIN0
#define STDOUT1
#define STDERR2

#define MAX_STRING 255

#define MAX_OUTPUT_WIDTH2448

#define START_GRAPHICS"\033G" /* use Control code group
"G" commands */
#define SEND_PLANE"\033I%c\03d"/* send image color line
*/

```

```

#define SEND_COMPRESSED"\033J%c"/* send compressed
image color line */
#define SEND_PRINT"\033A"/* End of this micro-line (4
planes) */

/* finish graphics mode [01] with a
2" gap or a FF */
#ifdef FORM_FEED
#define END_GRAPHICS "\014"
#else

#define END_GRAPHICS
"\012\012\012\012\012\012\012\012\012\012\012\012\012"
#endif

static char row_color[4][4] = {
    {'0', '4', '8', '<'},/* Row 1, Black, Magenta,
Yellow, Cyan */
    {'1', '5', '9', '='},/* Row 2, Black, Magenta,
Yellow, Cyan */
    {'2', '6', ':', '>'},/* Row 3, Black, Magenta,
Yellow, Cyan */
    {'3', '7', ';', '?'}/* Row 4, Black, Magenta,
Yellow, Cyan */
};

/*
** GLOBAL VARIABLES
*/

static STRING progname;
static UINT32 output_width=0;
static UINT32 output_height=0;

static UINT32height_loop;

/*
** MAIN
*/

int main(argc,argv)
int argc;
char*argv[];
{
    UINT32scanline[MAX_OUTPUT_WIDTH];
    UINT8*readline;
    int len,to_read;

```

```

    progname = argv[0];

    /*
    ** Read header information from pipe (width &
height)
    */
    {
        UINT32 header[2];
        if( read(STDIN, header, 8) != 8 ) {
            fprintf(stderr,"%s: Unable to header info.\n",
                progname,height_loop);
            exit(1);
        }
        output_width = header[0];
        output_height = header[1];
        if( (output_width < 1) || (output_width >
MAX_OUTPUT_WIDTH)
            || (output_height < 1) ) {
            fprintf(stderr,"%s: Bad width or height
info.\n",
                progname,height_loop);
            exit(2);
        }
    }

    (void) start_output();

    for( height_loop=0; height_loop<output_height;
height_loop++ ) {
        /*
        ** read the next scan line.  Because we are
reading from a pipe,
        ** we may have to do several reads to get the full
scan line
        */
        to_read = output_width*4;
        readline = (UINT8 *) scanline;
        while( to_read ) {
            len = read(STDIN, readline, to_read);
            if(len < 1) break;
            readline += len;
            to_read -= len;
        }
        if( len <= 0) {
            fprintf(stderr,"%s: Unable to read scan line
%d.\n",
                progname,height_loop);
            (void) finish_output();

```

```

        exit(3);
    }
    /*
    ** Process the scan line
    */
    (void) process_scanline(scanline);
}
(void) finish_output();
exit(0);
/*NOTREACHED*/
}
/*
** output() - output some data
*/
int output(data,length)
char *data;
int length;
{
    int wrote;
    if( length>0 ) {
        wrote=write(STDOUT,data,length);
        if(wrote != length) {
            fprintf(stderr,"%s: Tried to write %d bytes,
return=%d\n",
                progname,length,wrote);
            if( wrote<1 ){
                perror("Write error");
            }
            return(1);
        }
    }
    return(0);
}

/*
** start_output- sends init strings et al
*/
int start_output()
{
    output(START_GRAPHICS,strlen(START_GRAPHICS));
}

/*

** finish_output- sends shutdown strings et al
*/
int finish_output()
{
    output(SEND_PRINT,strlen(SEND_PRINT));
    output(END_GRAPHICS,strlen(END_GRAPHICS));
}

```



```

}

/*
** process_scanline- converts and outputs the scan line
*/

#define CYAN_ON0x1
#define MAGENTA_ON0x2
#define YELLOW_ON0x4

/* these must match the row_color table */
#define BLACK_PLANE0
#define MAGENTA_PLANE1
#define YELLOW_PLANE2
#define CYAN_PLANE3

static UINT8 dither[DITHER][DITHER] = {
    { 8, 136, 40, 168 },
    { 200, 72, 232, 104 },
    { 56, 184, 24, 162 },
    { 248, 120, 216, 88
};

int process_scanline(scanline)
UINT32 *scanline;
{
    static char current_row = 0; /* current row, from 0
to 3 */
    char send_plane[MAX_STRING];
    UINT32 pixel_loop, pixel;
    UINT32 plane_loop;
    UINT32 plane_byte; /* offset in plane_loop (bytes #)
*/
    UINT32 plane_bit; /* offset in byte of plane (bit #)
*/
    int vector_flag;

    UINT8 red_dither, green_dither, blue_dither;
    UINT8 turn_on, this_dither;

    UINT8 color_plane[4][ (MAX_OUTPUT_WIDTH+7)/8 ];
    /* Black, Magenta, Yellow, Cyan
planes */

    /*
** clear out the plane_buffer
*/
    for( plane_loop=0; plane_loop < (output_width+7)/8;
plane_loop++ ) {
        color_plane[0][plane_loop] =

```

```

        color_plane[1][plane_loop] =
        color_plane[2][plane_loop] =
        color_plane[3][plane_loop] = 0;
    }

    /*
    ** output each pixel in the scan line
    ** We work out the Red, Green and Blue dither values
then
    ** change to Cyan, Magenta, Yellow to output to the
printer
    */
    for( pixel_loop=0; pixel_loop<output_width;
pixel_loop++) {

        pixel = scanline[pixel_loop];
        /* get dither values. */
        red_dither = (pixel >> RED_SHIFT) & 0xff;
        green_dither= (pixel >> GREEN_SHIFT) & 0xff;
        blue_dither = (pixel >> BLUE_SHIFT) & 0xff;
        vector_flag= (pixel >> VECTOR_FLAG_SHIFT) & 0x01;
/* [02] */
        /*
        ** now convert dither patterns into colour
planes.
        ** We pull a trick here, by turning ON color
(e.g. YMC)
        ** if dither matrix is GREATER than value.
        */
        if(vector_flag) this_dither = 129;
        else this_dither = dither[pixel_loop %
DITHER][height_loop % DITHER];
        this_dither = dither[pixel_loop %
DITHER][height_loop % DITHER];
        if(red_dither < this_dither) turn_on = CYAN_ON;
        else turn_on = 0;
        if(green_dither < this_dither) turn_on |=
MAGENTA_ON;
        if(blue_dither < this_dither) turn_on |=
YELLOW_ON;

        plane_byte = pixel_loop>>3;
        plane_bit = (7 - (pixel_loop & 0x7));

        if(turn_on == (CYAN_ON | MAGENTA_ON | YELLOW_ON))
        { /* Black */
            color_plane[BLACK_PLANE][plane_byte] |= (1 <<
plane_bit);
        }
        else { /* Must be white or some colour */
            if(turn_on & CYAN_ON)

```

```

        color_plane[CYAN_PLANE][plane_byte] |= (1
<< plane_bit);
        if(turn_on & MAGENTA_ON)
            color_plane[MAGENTA_PLANE][plane_byte] |=
(1 << plane_bit);
        if(turn_on & YELLOW_ON)
            color_plane[YELLOW_PLANE][plane_byte] |=
(1 << plane_bit);
    }

}

/* [04] send image in RLL compressed mode */
for( plane_loop=0; plane_loop<4; plane_loop++ ) {
    UINT8 out_plane[MAX_OUTPUT_WIDTH];/* could be
much smaller; /8? */
    UINT8 *in_ptr = color_plane[plane_loop];
    UINT8 *diff_ptr = color_plane[plane_loop];
    int left = (output_width+7)/8;/* bytes left of
input */
    int run_count =0;/* count of rll bytes */
    int diff_count = 0;/* count of different bytes */
    int outcount = 0;/* count of output bytes */
    int notnull = 0;/* 1 if non all blank for this
line */
    int loop255;

    while(left--) {
        if(*in_ptr) notnull = 1;/* well, at least we
have some data */
        if(!left || (*in_ptr != in_ptr[1])) {
            if(run_count && (left || notnull)) {
                /* output the RLL data */
                run_count += 1;/* include this value */
                while(run_count > 255) {
                    out_plane[outcount++] = 'A'; /* t1:
rll flag */

                    out_plane[outcount++] = 0xff;
                    out_plane[outcount++] = *in_ptr;
                    run_count -= 255;
                }
                out_plane[outcount++] = 'A'; /* t1: rll
flag */
                out_plane[outcount++] = (UINT8)
run_count;
                out_plane[outcount++] = *in_ptr;

                run_count = 0;
            }
            else {

```

```

        if(diff_count==0) diff_ptr = in_ptr;
        diff_count++;
    }
}
if( !left || (*in_ptr == in_ptr[1])) {
    if(diff_count) {
        /* output the different data */
        while(diff_count > 255) {
            out_plane[outcount++] = 'B'; /* t2:
diff flag */
            out_plane[outcount++] = 0xff;
            /* a slow but portable loop - memcpy
would be faster */
            for(loop255=0; loop255<255;
loop255++)
                out_plane[outcount++] =
*diff_ptr++;
            diff_count -= 255;
        }
        out_plane[outcount++] = 'B'; /* t2:
diff flag */
        out_plane[outcount++] = (UINT8)
diff_count;
        /* a slow but portable loop - memcpy
would be faster */
        while(diff_count-->0)
            out_plane[outcount++] = *diff_ptr++;
        diff_count = 0;
    }
    run_count++;
}
in_ptr++;
}

if(outcount) {
    sprintf(send_plane,SEND_COMPRESSED,
            row_color[current_row][plane_loop]);
    output(send_plane,strlen(send_plane));
    out_plane[outcount++] = '@';/* t0: terminator
value */
    output(out_plane,outcount);
}
}

/***[04] this was the old non-compressed mode data

for( plane_loop=0; plane_loop<4; plane_loop++ ) {
    sprintf(send_plane,SEND_PLANE,

```

```

row_color[current_row][plane_loop], (output_width+7)/8)
;
    output(send_plane, strlen(send_plane));

output(color_plane[plane_loop], (output_width+7)/8);
}
***/
    current_row += 1;
    if(current_row > 3) {
        current_row = 0;
        output(SEND_PRINT, strlen(SEND_PRINT));
    }
}

```

The filter program takes the raw `ermhe` format and converts it to the hardcopy device specific format. Within the filter program you must allow for color compression, vector data, dithering and RLL and chroma correction if your hardcopy device can handle these.

## Color compression

The ER Mapper hardcopy engine output format is always a 24 bit RGB color image. Many hardcopy devices can only print a limited number of colors, in which case the number of colors has to be compressed from  $2^{24}$  (16.8 million colors).

By generating all `ermhe` output as 16 million color RGB and then compressing the image to suit the physical hardware it is possible to take advantage of whatever features the hardcopy device supports.

The compression routine in the above Sharp JX730 filter program maintains the most significant color differences with a technique known as ordered dithering.

Maintaining full RGB image quality until the last stage of output also allows output to lesser devices such as Grey Scale or limited color support and to devices using Hue Saturation Value instead of RGB.

## Dithering

Dithering is one of the tasks of the filter program. A range of 16 million value colors is available on the display but on many hardcopy output device these 16 million value colors are not possible. Without dithering, most color printers have only 8 colors per pixel available, made up of a mixture of cyan, magenta, yellow and black.

Dithering is used to overcome this limitation. Some hardcopy filter programs provided with ER Mapper use an ordered dithering algorithm which is included as an example in the Sharp JX730 hardcopy filter program earlier in this chapter. Ordered dithering increases the color range and reduces artefacts otherwise created by other dithering algorithms such as random dithering.

Dithering accommodates the continuous shade variations which are necessary in remotely sensed images to produce less sharp boundaries between the colors. This increased color range is at the expense of spatial resolution when the image is changing color density rapidly.

If vectors are dithered, the vectors on hardcopy look broken up and blur into the background of the image. The filter program can check if the pixel was a vector before rasterization, by checking the vector flag. If the vector flag is set, then the filter program can decide not to dither this pixel and choose one of the eight primary colors instead.

One disadvantage of not dithering vectors is if you had asked for a pink vector then you are probably going to get a red colored vector. An advantage of dithering is that the vectors will be sharp and distinguishable from the background raster image data.

Dithering is not mandatory, but is used in many of the supplied ER Mapper hardcopy filter programs.

The ER Mapper hardcopy filter programs supplied use different dithering algorithms such as Ordered Dithering, Error Diffusion, Floyd Steinberg etc.

Hardcopy devices other than color printers, such as filmwriters do not need to use dithering.

## **Image compression**

Because images generated can be very large, up to several hundred megabytes, many hardcopy devices support compression techniques such as run length limited (RLL). The filter program performs any image compression required for the hardcopy. In the example Sharp JX730 filter program the RLL compression technique is used.

## **Chroma correction**

If the printer documentation includes an algorithm for chroma correction this can be implemented in the filter program.

## **Output Program**

The output program takes the hardcopy device specific format and directs it to a printer or file.

To direct the output to a local printer, use the following statement:

```
"he_writetofile :LPT1"
```

The output will go to the printer connected to local port LPT1.

You can also print to a file using the following statement:

```
"he_writetofile C:\documents\printfile.prn"
```

## Subsampling/Supersampling

Subsampling and supersampling are automatically performed by the ermhe hardcopy engine. The output images are always generated at the resolution of the hardcopy device regardless of the resolution of the input images used in the algorithm. These means the quality of the hardcopy is always as high as can be physically provided by the hardcopy device.

## Aspect ratio

The aspect ratio for the hardcopy is the same as for the algorithm on screen. A 1:1 aspect ratio is enforced by default.

## Hardcopy output size

Hardcopy output size can be specified in three way: full resolution of hardcopy device, measurement in inches or metres and scale.

To use the full resolution of the hardcopy device do not specify the size or the scale of the hardcopy. The result is hardcopy which is one page in width.

Secondly, you can specify how large you would like the output in inches or metres. ER Mapper will automatically strip print if the size is greater than that of the paper type in the hardcopy device.

In the third case, you can specify a scale. ER Mapper uses the cell size of one of the images to scale the hardcopy produced.

In each method ER Mapper works out how large the image has to be and calculates and prints as many strips as required.



---

*You can use the ER Mapper Page Setup Wizard or Dialog to set up the page sizes. Refer to the [Page setup](#) chapter in the ER Mapper User Guide.*

## Color

The colors produced by the hardcopy device and those colors shown on the display are essentially the same. However, hardcopy devices use the subtractive color system whereas the display uses the additive color space system. It is difficult to create hardcopy exactly as it appears on the display because of how our eyes perceive the different color spaces and the fact that the vectors are rasterized.

The ER Mapper display background is set to black and creates colors by adding colored light to the black background. Small changes in intensity in the dark shades are perceived more readily than in the light shades. On the white background of the hardcopy device, ink is added to a white background to produce various colors, that is they are "subtracted" from white.





# Importing using a command line

The most common way to import an image is to use the **Import** commands on the ER Mapper **Utilities** menu. However, you can also use command line options detailed in this chapter.

If you are to use the import utilities on a command line basis, rather than the ER Mapper user interface, it is easiest to use them in the directory where the imported file will be situated. Similarly, 'mergeers' and 'invert', should be used in the directory which contains the input image files.



---

See *mergeers* and *invert* later in this chapter.

Before discussing the import programs some background information on ER Mapper data storage method is required.

## Raster and vector image formats

The two data types used by ER Mapper are raster and vector. In each case the data is stored in a file and has an appropriate ASCII header file which describes the image more fully. The header file for raster data has a file extension of **.ers**. The extension for vector header files is **.erv**. An example raster image **'my\_data\_set'** consists of two files named **'my\_data\_set'** and **'my\_data\_set.ers'**.

The data file itself is a binary file arranged as a Band Interleaved by Line (BIL) image file. A BIL data file contains all cells for a single image line as a sequential group of bands. To illustrate, if a BIL file had two lines (L) of three cells (C) and contained two bands (B) the sequence of values in the file would be:

- L1B1C1, L1B1C2, L1B1C3,
- L1B2C1, L1B2C2, L1B2C3,
- L2B1C1, L2B1C2, L2B1C3,
- L2B2C1, L2B3C2, L2B2C3

where L1B1C1 represents Line 1, Band 1, Cell 1 etc.

The data import utilities are used to translate raster or vector data from one format to a format recognised by ER Mapper and to create a header file for the data.

The ER Mapper import utilities include utilities for importing raster header files from many of the sensing platforms (see *Appendix A, "Sensor platform characteristics"*). These files provide sensor characteristics and are used as a template by the import utilities. Header information is copied from the sensor files and is combined with the basic information extracted from the tape or specified at the utility command prompt. In the case of raster files the basic information includes the number of image lines, and the number of cells per line. The resultant header file contains additional information about the image which, although not essential for data display, is highly desirable.

The sensor files may be found in the `sensortype` directory. Use the supplied sensor files to assist you in creating your own additional sensor files.



---

*If you are experiencing difficulty importing text data files (e.g. `importascii`, `importgeoimage`), check that the transfer of the original raw text data file to the current machine was done using a text mode of transfer. Transfer of text data files using a binary mode of transfer instead of text mode may result in the file being corrupted with control characters. This manifests itself when copying text files from PC DOS/MSDOS to Unix.*

## Raster File Import Switches

The import programs may be grouped into raster file import and vector file import utilities. The raster import programs are discussed in this section.



---

*Vector import programs are discussed in the section ["Vector file import switches"](#) later in this chapter.*

The use of the group of raster import utilities are similar. They all share common option switches. The syntax of the raster import programs is as follows:

```
erm_run importprogram [switches] source_image
header_file.ers
```

When running command line programs you must prefix the command with "erm\_run". "erm\_run" sets up the execution path so that all the executable components can be found, and sets up any environmental variables. This is done automatically when you use the import utilities from inside ER Mapper.

The data importation is handled transparently by the import program. The switches are:

- `-?`: Displays a help message outlining the available switches. general usage message. For example,

```
erm_run importascii -?
```

would display the general usage and switches for the importascii import program.

- **-t**: Table of Contents. Shows details about the image without importing the data. This switch allows the contents of a tape to be examined before using the import program to actually import the data.
- **-q**: Query. Displays information about the file being imported and prompts you for acceptance of the file before proceeding with the import. This is useful when you are uncertain that the tape you are using contains the correct file. If you accept the file data importing continues normally.
- **-v**Verbose. Displays messages during the importing. Progress reports are displayed as each line is read and processed. If additional -v switches are used additional information is provided.
- **-l <line\_range>**: Specifies a sub section of the image lines to be imported. For example,

```
-l20-300
```

will import lines 20 to 300 inclusive. The default setting for the line range is all lines of the image.

When specifying lines with the -l switch the first line is zero, and start and end lines are included, for example, to import the first 5 lines of an image,

```
-l0-4
```

- **-c <cell\_range>**: Specifies a sub section of the columns of the image to be imported. It can be combined with the -l switch above. For example,

```
-c100-6000
```

will import columns 100 to 6000 inclusive into the image. The default range is all cells of the image.

When specifying cells with the -c switch the first cell is zero, and start and end cells are included, for example, to import the first 5 cells of an image,

```
-c0-4
```

- **-b <band\_range>**: Specifies the bands to be imported. The bands may be listed as band numbers separated by commas (1,2,3,6,..) or groups of bands within a range inclusively (1-3) or a combination of the two methods (1-3,7,9,11-14,..). The default band selection is all bands.

When specifying a range of bands with the -b switch the first band is band 1, and start and end bands are included.

- **-s <sensor\_type>**: The selection of a sensor type allows the import utility to read the appropriate `.ers` file for the sensor. This contains a template for information such as wavelengths and bandwidths for the sensor channels, cell dimensions etc. The header file created during the importation will contain additional information as was available from the `sensor_type.ers` file. If a user specified sensor name starts with `.\` or `\` the path indicated will be searched, otherwise ER Mapper will search the path `sensortype` for the `sensor_type.ers` file.
- **-f <file\_skip>**: This allows files on the input device to be skipped in order to reach the file required for importation. This switch is only supported when importing from devices such as magnetic tape drives that are capable of skipping files. A non-rewind device, such as `/dev/nrs6250t0`, must be used.
- **-x**: Use the user interface instead of the command line interface. The import program user interface window will be displayed prompting for information about the image, which would have normally been specified using other switches on the command line. For example,

```
$ erm_run importascii -x
```

will display the importascii user interface window.

The import program user interface is also accessible by choosing **Import Raster** or **Import Vector** options from the ER Mapper **Utilities** menu.




---

See the [Importing and exporting data](#) chapter in the ER Mapper User Guide.

- **-P <name>**: Specifies the projection of the image.
- **-D <name>**: Specifies the datum of the image.
- **-r <value>**: Specifies the rotation of the image from North, in decimal degrees anticlockwise.

## Importing Raster File examples

### Example 1

```
$ erm_run importascii -f3 -t /dev/rst17
```

will skip to the third file on a 6250 bits per inch (bpi) CCT and display the details about the file. No data will be imported. Remember that file skipping on a CCT requires a non-rewind device; in this case `/dev/rst17`.

### Example 2

```
$ erm_run importascii my_ascii_file my_image.ers
```

imports from disk rather than from a CCT. The source file "**my\_ascii\_file**" is read and imported into ER Mapper format. The specified destination file must have the **.ers** file extension. ER Mapper creates two files; the data file (**my\_image**) and the header file (**my\_image.ers**).

### Example 3

When using `importbil` you must specify the number of bands, number of lines and number of cells in the image. The program will ask you to enter these values. There is no standard header under BIL format, because of the large number of tape formats used. For example, if we wanted to import a file from a CCT device **/dev/rs6250** a BIL format file of 512 lines by 512 cells by 2 bands:

```
$ erm_run importbil /dev/nrs6250t0 bil.ers
```

The program would then prompt for the lines, cells and bands as follows:

```
Please enter # of lines (rows) in input image : 512
#lines: 512
Please enter # of cells (columns) in input image : 512
#cells: 512
Please enter # of bands (channels) in input image : 2
#bands: 2
```

In the above example, the user has entered the number of lines, cells and bands when prompted by the `importbil` program.

### Example 4

```
$ erm_run importcct /dev/rst17 my_image.ers
```

Loads the first file on the non-rewind CCT device into the ER Mapper image file **my\_image**. The header file **my\_image.ers** will contain a specification of the file size (lines, pixels, and bands). Note that the `importcct` program will only accept a tape drive as the data source. Unlike `importcct`, the other raster import programs can accept data from other accessible devices; hard disk and optical disk storage and files located on other computers (if ER Mapper is connected to a network).

### Example 5

Copy a small subsection and single band of a DISIMP format file on the 1/4 inch cartridge tape device **/dev/rst8** to an ER Mapper raster file **postage\_stamp**; **postage\_stamp.ers** will be created during the import. The first file on the tape will be copied.

```
$ erm_run importdisimp -l20-50 -c400-430 -b2 /dev/rst8
postage_stamp.ers
```

### Example 6

Subsection a square image (1024 lines and 1024 cells) of all bands from the disk file **dolaimage** into the ER Mapper file **test\_zone**. Display the file details and request confirmation that the file is the correct one before copying the data.

```
$ erm_run importdola -l1-1024 -c101-1124 -q dolaimage
test_zone.ers
```

### Example 7 (importusgs)

Import the disk file `'sample_data'` in USGS format into the ER Mapper file `'my_data.ers'`. ER Mapper will ask if the requested file is the correct one (-q) and will print a progress report as each line is copied (-v).

```
$ erm_run importusgs -q -v sample_data my_data.ers
```

## Vector File Import Switches

The vector import utilities are used in a similar way to the raster image data utilities discussed above. The type of data provided by vector files makes several of the option switches used for raster files inappropriate.

The syntax of the vector import programs is as follows:

```
erm_run importprogram [switches] source_image  
header_file.erv
```

The data importation is handled transparently by the import program. The switches are:

- **-?**: Displays a help message, outlining the switches.

```
erm_run importas2482 -?
```

would display the general usage and switches for the `importa2482` import program.

- **-t**: Shows a Table of Contents with details about the image being processed by the import program. It examines the image but does not actually import the data. Used to examine a tape before using the import program to actually import the data.
- **-q**: Query. Displays information about the file being imported and prompts you for acceptance of the file before proceeding with the import. This is useful when you are uncertain that the tape you are using contains the correct file. If you accept the file data importation continues normally.
- **-v**: Verbose. Displays progress reports as each line is read and processed. If additional -v switches are used additional information is provided.
- **-f <file\_skip>**: Skips files in order to reach the file required for importation. This switch is only supported when importing from devices such as magnetic tape drives that are capable of skipping files. A non-rewind device, such as `/dev/nrs6250t0`, must be used.
- **-x** Allows the user interface to be used instead of the command line interface. The import program user interface window will be displayed, prompting for information about the image which would have normally been specified using other switches on the command line. For example,

```
erm_run importas2482 -x
```

will display the importascii user interface window.

The import program user interface is also accessible by choosing import options from the ER Mapper **Utilities** menu.



---

See the [Importing and exporting data](#) chapter in the ER Mapper User Guide.

- **-P <name>**: Specifies the projection of the image.
- **-D <name>**: Specifies the datum of the image.
- **-r <name>**: Specifies the rotation of the image from true North, in decimal degrees, counterclockwise.



---

The switches listed above are all common to the vector data import utilities. The vector import formats currently supported are listed in the [Supported import formats](#) chapter in the ER Mapper User Guide.

## Importing Vector File examples

### Example 1

```
$ ermrun importas2482 /dev/nrs6250t0 mainroads.erv
```

Imports the AS2482 format vector file on the half inch magnetic tape into the ER Mapper vector file 'mainroads'. The vector header file '**mainroads.erv**' will be created automatically.

### Example 2

```
$ erm_run importgeoimage -f3 -t /dev/rst17
```

Shows details for the GeoImage format vector file which is the third file on the half inch magnetic tape **/dev/rst17**. The file is not imported.

## Invert

The 'Invert' utility is used to change the orientation of either an **.ers** raster file or a virtual dataset. It can be run from the command line using the switches described below.

```
erm_invert [-vhbwx] input_image output_image
```

For example to invert an image from left to right and top to bottom:

```
invert -b input_image inverted_image
```

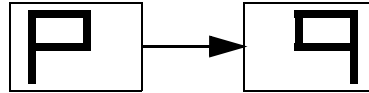


---

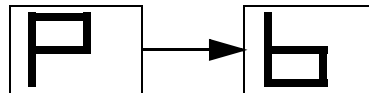
The inversion process treats the data as raw data. It will not invert your coordinate system.

The switches are described below. You can use the `-x` flag to run the program with the graphical interface, in the same way as though you had chosen the **Invert a Raster or Virtual Dataset** option from the **Utilities / File Maintenance / Datasets** menu. If you don't use the `-x` flag, you must use one of the `-vhb` flags and give the input and output image names.

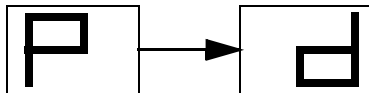
- `-v`: Inverts the data around the vertical line of symmetry.



- `-h`: Inverts around the horizontal line of symmetry.



- `-b`: Inverts around the horizontal and vertical line of symmetry.



- `-w`: Overwrites an existing output file.
- `-x`: Runs the program using the graphical user interface.
- **input\_image**: The input image can either be a full directory path or a path relative to the current path.
- **output\_image**: The output image name needs to include the directory path, otherwise it will be output into the same directory from which you ran `erm_invert`.



---

*Some of these facilities are also available on the ER Mapper main dialog box. Select **File Maintenance / Datasets** from the **Utilities** menu.*



# Utilities

There are a number of utility programs that are used in various places by ER Mapper. They are included here because you may find them useful to incorporate in your own dynamic links options.

## Coordinate conversion

### **togeo**

This command line utility prompts the user for rectangular (EN) coordinates of a point in space and outputs the corresponding LL coordinates based on the projection and datum entered by the user. All information needed by the utility is acquired during the program run and hence, no arguments are required on the command-line.

### **fromgeo**

This utility is the inverse of "togeo". It takes the LL coordinates of a point and outputs the corresponding rectangular (i.e. EN) coordinates based on the projection and datum entered. As with "togeo" all information needed by the utility is acquired during the program run itself and not from the command-line.

### **gdt\_conv**

Similar to the above conversion routines. It prompts the user to enter an input projection/datum pair and an output projection/datum pair than a 3-D coordinate. The utility then converts the entered coordinate to the corresponding coordinate of the point in the new (i.e. output) projection/datum coordinate space.

## Tape utilities

### **erm\_tapeutil**

This utility mimics the behavior of the UNIX system command "mt". It allows the user to perform basic tape operations like rewinding tapes, skipping files on a tape, checking for the current status of a given tape device etc.

- **Syntax:** `erm_tapeutil -f <tape_device> <command_to_issue>`
- **<tape\_device>**: The name of the tape device for which the command is intended
- **<command\_to\_issue>**: One of the following:
  - bof: Finds the beginning of the current file on the tape
  - dump [n]: Reads a record from the file and dump [n] characters.



---

*The default number of characters read equals the size of the record to be read.*

- fsf [n]: Skips past "n" files on the tape.

## tape\_struct

- rew: Rewinds the tape to the beginning.

This utility creates a report on the structure of data provided on tape and it typically used for determining the exact origin/format of a given tape-based image.

- **Syntax: tape\_struct <tape\_device>**
- **<tape\_device>**: The name of the tape device.

## dump\_cct

This utility provides the same functionality as that provided by "tape\_struct" above.

- **Syntax: dump\_cct <tape\_device>**
- **<tape\_device>**: The name of the tape device.

## tape2disk

This command line utility reads all files from tape into the specified disk file maintaining the original structure of the data (i.e. with EOF markers and so on).

- **Syntax: tape2disk <tape\_device> <output\_disk\_file>**
- **<tape\_device>**: The name of the tape device.
- **<output\_disk\_file>**: The name of the target disk file.

## disk2tape

This utility performs the inverse operation to the above. It takes a disk file whose structure is identical to that of the original data on tape and recreates the structure on the tape device specified as the second argument to the program.

- **Syntax: disk2tape <disk\_file\_name> <output\_tape\_device>**
- **<disk\_file\_name>**: The name of the disk file containing the data.
- **<output\_tape\_device>**: The name of the target tape device.



---

*The above two utilities are typically used together for copying tapes; e.g. creating a low-density copy of a high-density tape.*

## Image file compression

### ecw\_compress

Use this command to compress an image file size. As this is an ER Mapper executable, you need to ensure that the ER Mapper environment variables are correctly set up; the easiest way to do this is to use the "erm\_run" command. "erm\_run" sets up the execution path so that all the executable components can be found, and sets up any environmental variables.

#### Syntax:

```
erw_compress infile [-show] [-o outfile] [-c  
compression_ratio] [-g | -rgb | -multi]
```

- **-show:** report the expected output size only, and not compress
- **-nowait:** after compression, do not report a message dialog and wait for user
- **-g:** compress to a single band UINT8 greyscale output file.
- **-rgb:** compress to a RGB file optimized for color imagery compression.
- **-multi:** will compress to a multi-band image (only 3 band, UINT8 format, supported at present)

#### Example:

```
C:\Program Files\ERDAS\ERDAS ER Mapper  
7.2\bin\win32\erm_run erw_compress input.ers -o  
output.ers -rgb -c 20
```

### **ecw\_compress\_gui**

Use this system command in your ER Mapper wizard batch scripts to interface to the compression engine.

#### Syntax:

```
erw_compress_gui infile [-show] [-o outfile] [-c  
compression_ratio] [-g | -rgb | -multi]
```

- **-show:** report the expected output size only, and not compress
- **-nowait:** after compression, do not report a message dialog and wait for user
- **-g:** compress to a single band UINT8 greyscale output file.
- **-rgb:** compress to a RGB file optimized for color imagery compression.
- **-multi:** will compress to a multi-band image (only 3 band, UINT8 format, supported at present)

#### Example:

```
$compress_command = "ecw_compress_gui \" + $FromFile +  
\" \" -o \" + $OutputFile + \" \" -c \" + $TheRatio + \" \" +  
$compress_format  
system $compress_command
```



# Batch scripting and wizards

This chapter describes how you use the ER Mapper scripting language to create batch scripts. These batch scripts can range from a list of operations which can be run from a single command line to dialog boxes and wizards with which users can interact.

You can connect these batch operations to the menus and toolbars on the ER Mapper user interface.

The scripting language includes powerful commands for creating wizards. ER Mapper is supplied with a number of pre-designed wizards accessible via the menus and toolbars. You can use the scripting language to customize these or create your own. Refer to the ["Wizard scripts"](#) section for information on creating wizards.



---

*You can design wizards so that they "remember" settings or values entered. See ["Using preferences to remember settings"](#).*

## Batch script documentation

Additional documentation on batch scripting may be found in the following chapters:

The ["Scripting language"](#) chapter covers the structure and composition of the scripting language. It describes data input and output methods and gives an overview of the actions that can be performed on the different ER Mapper objects. It also describes the structure of the commands.

The ["Scripting reference"](#) chapter lists and describes all the Keywords, Operators, Variables and Commands used by the scripting language. The Command descriptions include their syntax and examples of their use. The chapter contains the following sections:

Section	Description
"Operators"	Suite of standard operators
"Mathematical functions"	Mathematical functions
"Variables"	Variable types and their allowed values
"Page size options"	Standard page sizes that can be used
"Keywords"	List of keywords
"Flow control"	Flow control components like labels and 'goto' commands.
"Including files"	How to include library batch files

"Error reporting"	Reporting errors
"Script commands - alphabetical listing"	List of all the script commands in alphabetical order.

## Creating a batch script

The steps involved in creating your own batch script are as follows:

1. Design the layout of dialog boxes if they are being used.

Decide on the appearance of dialog boxes, keeping in mind current conventions. We suggest that you maintain the look and feel of the existing ER Mapper dialog boxes. If you are designing a wizard, draw a flow diagram showing the sequence of the dialog boxes that will be displayed depending on the user inputs.

2. Use a text editor to create the script.

You can use any text editor to create the script. It may be easier to use the **Utilities / Batch Scripts / Create a Batch Script** menu option. This will save the new batch script in the \batch directory and open it in a text editor.

Browse through the 'batch\lib' directory to see if there are any library scripts you can include in your script.

It might also be easier to edit an existing batch script using the **Utilities / Batch Scripts / Edit a Batch Script** menu option, and then save it with a new name.




---

Refer to the ["Scripting reference"](#) chapter for information on the syntax of the commands.

3. If required, edit the menu (.erm) and toolbar (.bar) files to make the batch program accessible via the menus and/or toolbars.

Use either the **Utilities / Toolbars / Create a Toolbar** or the **Utilities / Toolbars / Edit a Toolbar** menu option depending on whether you want to create a new toolbar or to add an icon to an existing toolbar.

Use the **Utilities / User Menu / Edit a File** menu option to add a new entry to the ER Mapper menus.




---

Refer to the ["Menu and toolbar files \(.erm\) and \(.bar\)"](#) chapter for information editing menu and toolbar files.

## Getting started

The following procedure describes how you would;

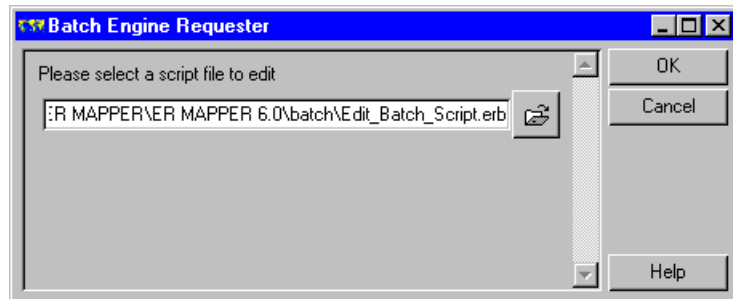
- Create a simple "Hello World" batch script
- Insert a menu option to run your script in the Process menu.

- Add a button to run your script to the Standard toolbar.

### Create a simple batch script

1. Select **Batch Scripts / Edit a Batch Script** from the **Utilities** Menu.

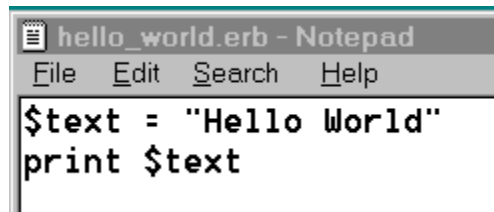
The Batch Engine Requester dialog box opens.



2. Enter "hello\_world.erb" for the name of your new batch file, and click on the **OK** button.

ER Mapper opens the text editor specified by the **ERMEDIT** environment variable. If you have not set this for a specific editor, it will default to NotePad.

3. Enter the following text in the editor window. Save the file and exit the editor.



4. Click on the **Close** button on the **Batch Engine Output** window if it is open.

### Insert a menu option entry

1. Open the file **config\processmenu.erm** in your text editor.
2. Add the following lines to the processmenu.erm file:
3. Save the modified processmenu.erm file.

### Add a button to the toolbar

You will first need to create a 24-bit 16 x 16 pixel tiff file for the button and copy it into the 'icons' directory.

1. Using any raster graphics application such as Corel PhotoPaint or PaintShopPro, create an appropriate Image. Save it as a 16x16 pixel tiff (.tif) file, **Hello.tif**, in the 'icons' directory.

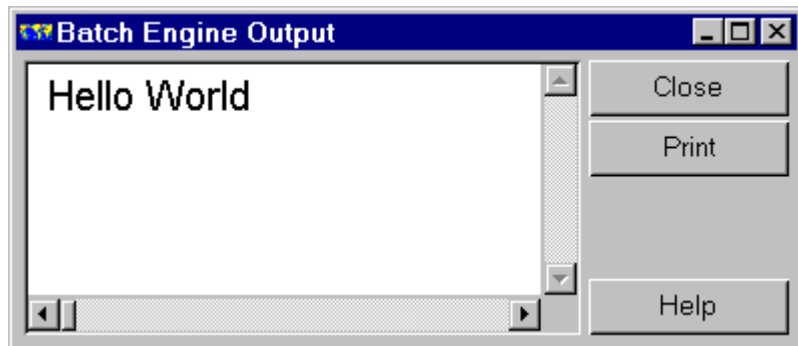


*You can also use ER Mapper to save an image as 16x16 pixel tiff file.*

1. Open the file **config\Standard.bar** in your text editor.
2. Add the following lines to the Standard.bar file.
3. Save the modified Standard.bar file.

### Run the batch script

1. Exit ER Mapper and restart it.  
The new toolbar button should now be visible on the **Standard** toolbar.
2. Move your mouse pointer over the new button. Check that "Hello World" is displayed as a tooltip.
3. Click on the "Hello World" button to run the script.  
The Batch Engine Output dialog should open displaying the words "**Hello World**".



4. Click on the **Close** button to close the **Batch Engine Output** dialog.
5. Select the **Process** menu and note that a **Hello World** option has been added.
6. Select **Hello World** from the **Process** menu and check that the batch script runs as before.

### Run the batch script from a command file

It is also possible to run a batch program from a command line outside of ER Mapper using the **ermapper -b** command; e.g:

```
ermapper -b batch_script
```



where `batch_script` is the name of your batch program.  
To run your script from a command line you must add the following line to the script:

```
exit n
```

Where *n* is the exit code returned from ER Mapper.

- *n* = 0: exits the script and leaves ER Mapper open.
- *n* = >0: exits the script and closes ER Mapper.

1. Open an MS-DOS Prompt window.
2. In the MS\_DOS Prompt window, change to the '**batch**' directory.
3. Enter the following command:

```
ermapper -b hello_world
```

This should run the script as before.

You could also have run the script from any directory using the full path name, e.g.:

```
ermapper -b "c:\Program Files\ERDAS\ERDAS ER Mapper  
7.2\batch\hello_world"
```



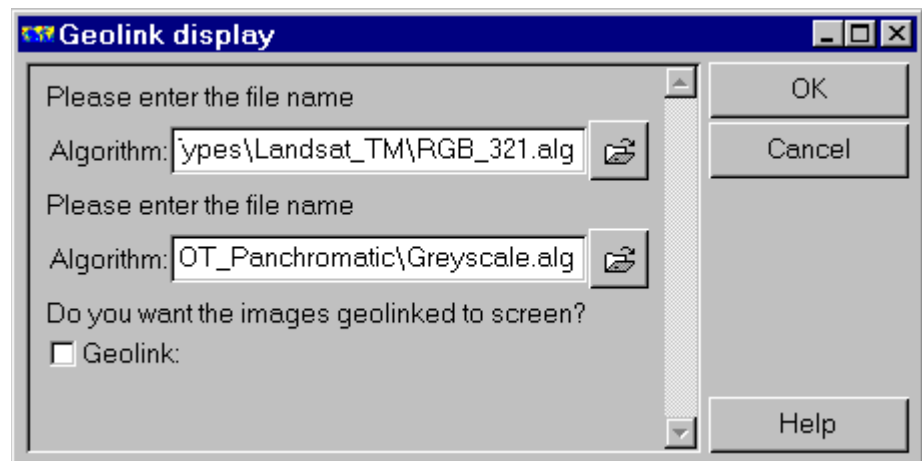
*Make sure that your PC PATH environment is set up correctly so that it can locate the `ermapper.exe` file in the '`bin\win32`' directory.*

### Example display and geolinking script

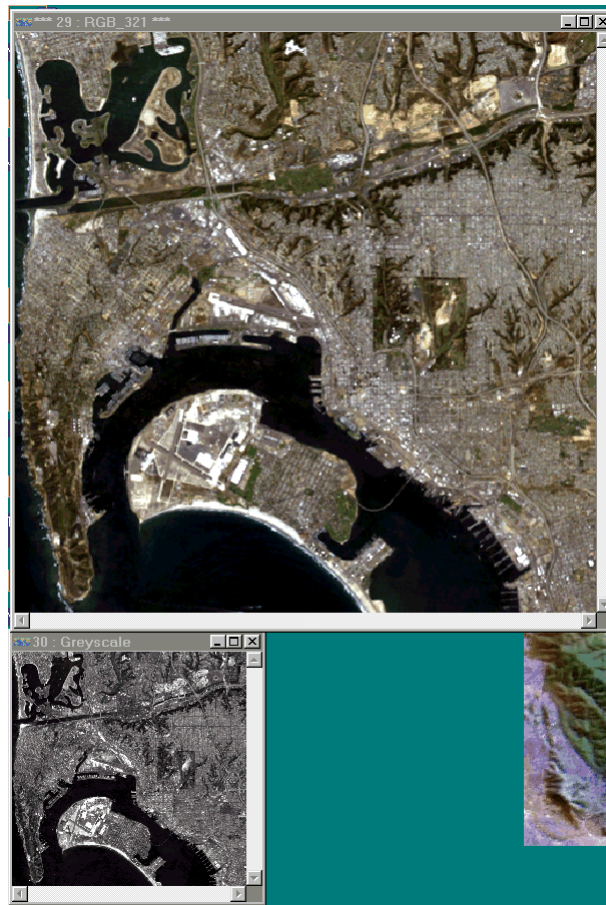
Now that we have managed to write and execute a "Hello World" script we can try something more challenging.

This example batch script will perform the following tasks

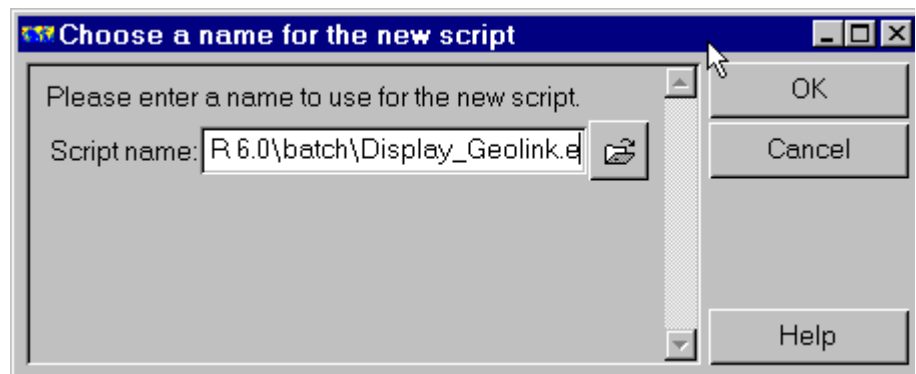
- Open a dialog box which allows the user to choose or enter the filenames of two algorithms to be displayed. The user is also given the option of having the algorithm display windows SCREEN geolinked.



- Display the two algorithms in two separate windows, one large and one small. The large window will be positioned in the top left corner of the screen, with the small window just below it.



1. Select **Batch Scripts / Create a Batch Script** from the **Utilities** menu.
2. Enter 'Display\_Geolink.erb' in the **Script name:** field of the **Choose a name for the new script** dialog.



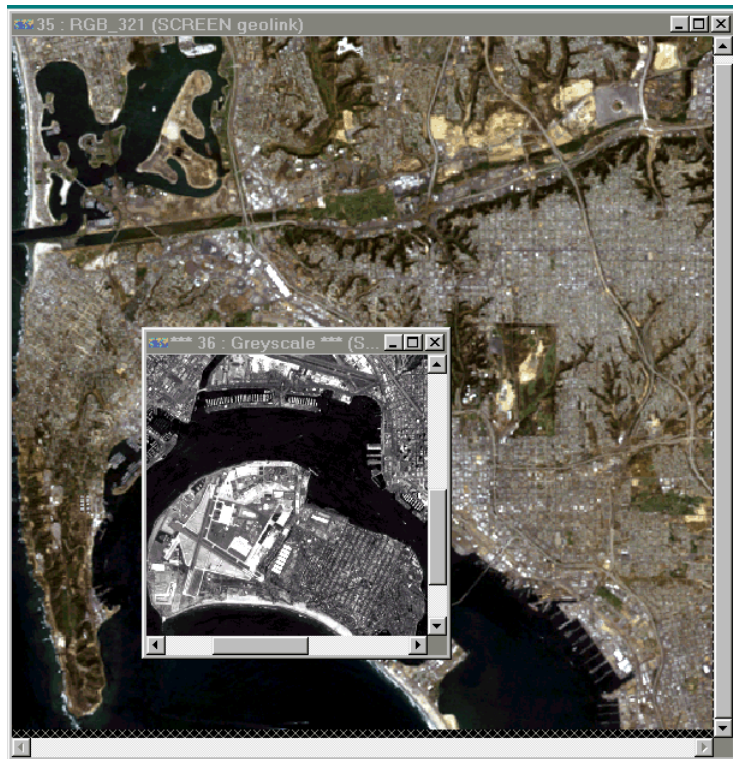
3. Enter the following script in the text editor page that opens:

Script	Comments
ask begin	Start of dialog box. Must have a corresponding 'ask end' later.
title "Geolink display"	Title of the dialog box
\$count = 0	Initialize the loop by setting the count to 0.
increment:	Label for loop re-iteration
say "Please enter the file name"	Optional text in the dialog box.
\$myfile[\$count] = ""	Initializes the \$myfile[] variable
ask file "Algorithm:" ".alg" \$myfile[\$count]	Add <b>Algorithm</b> field to dialog box for user to enter an algorithm filename with a .alg extension. The field includes a button to open the ER Mapper file chooser. The filename is stored in the \$myfile[] variable.
\$count = \$count + 1	Increment the count
if (\$count == 1) then goto increment	Repeat the loop until condition is satisfied. (In this case the loop will be repeated once)
\$link = 0	Initialize the yesno \$link variable to 0 (No).
say "Do you want the images geolinked to screen?"	Optional text in the dialog box.
ask yesno "Geolink:" \$link	Add selection box for users to check if they want the windows to be geolinked and thus set \$link to 1 (Yes).
ask end	End of dialog box.
\$win[0] = new window 0 0 600 600	Add new 600 x 600 pixel image window situated in the top left corner of the screen (0 0). Set \$win[0] variable to point to this window.
\$win[1] = new window 0 600 250 250	Add new 250 x 250 pixel image window situated 600 pixels vertically below the top left corner of the screen (0 600). Set \$win[1] variable to point to this window.
\$count = 0	Initialize the loop by setting the count to 0.
increase:	Label for loop re-iteration.
select \$win[\$count]	Select \$win[\$count] to be the active window.

Script	Comments
load algorithm \$myfile[\$count]	Load the algorithm whose filename is stored in \$myfile[\$count] into the batch engine.
copy algorithm to window	Copy the loaded algorithm from the batch engine to the active window.
if (\$link != "yes") then goto no_link	Check if the user selected Yes for geolinking the windows (\$link = "yes"). If not then skip the next line and go to the no-link label.
set window geolink mode to screen	Set the currently active window to geolink SCREEN mode.
no_link:	Label for 'if ..then goto..' command.
\$count = \$count +1	Increment the counter for the loop.
if (\$count == 1) then goto increase	Repeat the loop if the condition is met.
end_script	End of script label.

4. Save the Display\_Geolink script and exit the text editor.
5. Add the **Display\_Geolink** menu option to the **Process** menu by editing the processmenu.erm file. See ["Insert a menu option entry"](#) above for an example of doing this.
6. Add a **Display\_Geolink** button to the **Standard** toolbar by editing the Standard.bar file. ["Insert a menu option entry"](#) above for an example on doing this.
7. Run the script by either selecting **Display\_Geolink** from the **Process** menu, or by clicking on the **Display\_Geolink** button.

To try this script, we suggest that you use the ``\examples\Data_Types\Landsat_TM\RGB_321.alg'` in the large window, and ``examples\Data_Types\SPOT_Panchromatic\greyscale.alg'` in the small window. Run the script with and without selecting the geolinking option. When select the geolinking option, move the small window over the large widow to view the results. The expected result is shown below:



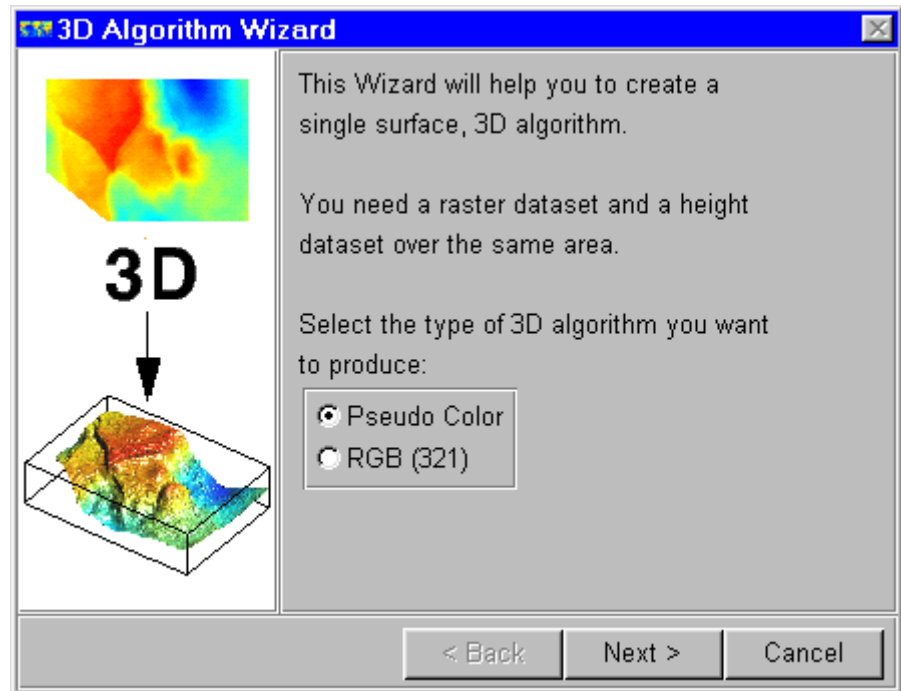
## Wizard scripts

The ER Mapper batch script language enables you to create interactive guided wizard scripts. Wizards are made up of one or (usually) more pages that step a user through a task. They guide the user to making choices, providing any necessary information and suggesting appropriate choices when possible. They are especially useful for complex tasks or tasks requiring a number of steps.



---

*For information about general batch script commands see the [“Scripting language”](#) chapter.*



### Example wizard script

You can insert batch commands to create a wizard interface into any batch script. The structure of the script will vary depending on the nature of the task but will generally be made up of the following sections.

- a setup section in which variables and default values are set up and any necessary information is copied from the current algorithm or elsewhere
- a wizard section in which the layout and contents of the wizard pages are specified, as well as where to store any user input and any procedures to be run as a consequence of the user's choices
- an outcome section in which the procedures to be carried out as a consequence of the user's choices in the wizard are specified.

This chapter is concerned with the wizard section.

The simple wizard script 'Image\_View.erb' is shown next.

```
# Script: Image_View.erb
# Summary: Wizard to view an image
# Details: Wizard wrapper around Create_RGB.erb
# Creates an RGB algorithm from the current window
algorithm.
# It uses the following strategy:
#1)Look for any valid RGB layers.If any are present,
turn them
# on, run & exit.
```

```

#2) Check the current layer - if it has a dataset it
uses it to
# create the algorithm (simply turning the layer on if
it's a
# pseudocolor layer).
#3) Look for an active pseudocolor layer. If one is
found, use
# its dataset to create the RGB algorithm.
#4) Look for any raster layer with a valid dataset. Use
it to
# create the new algorithm.
#5) At this point there is no valid dataset, so ask for
one &
# use it to create the RGB algorithm.

include "lib/BE_Startup.erb"
if current window then goto window_ok
new window

window_ok:
copy algorithm from window

# Set the layer to it
$LIB_lay1 = current layer
$LIB_alg1 = current algorithm
$LIB_tra1 = last transform
$image_filename = get layer dataset
-----
-----

```

Wizard pages are defined using the following format.

```

-----
-----
include "lib/BE_Startup.erb"#wizard set up

WizardPage1:          #label for page 1
WizardPage begin "WizardName"#title for first page
  title "Page1Name" #if a new wizard
  container begin "Image"
    container information#usually image info
    ...                #on first page
  container end
  container begin "DataEntry"
    container information
    ...
  container end
  container begin "PageControls"#control buttons
    ask action "< Back"
    ask action "Next >" goto WizardPage2
    ask action "Cancel" close

```

```

        container end
WizardPage end

WizardPage2:          #label for page 2
WizardPage begin     #no title
    title "Page2Name"
    container begin "container1name"
        container information
        ...
    container end
    container begin "container2name"
        container information
        ...
    container end
    container begin "container3name"
        container information
        ...
    container end
WizardPage end

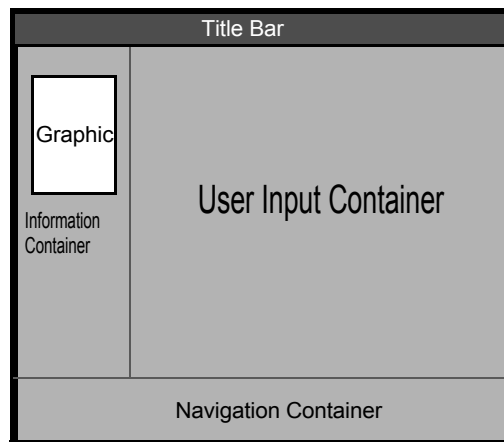
```

---



---

## The layout of wizard pages



- Wizard pages are made up of a number of different areas. The exact design of wizard pages varies but generally the pages consist of
- an information container at the left which usually has a graphic image.
- a user input container to the right
- navigation container with buttons at the bottom
- graphics can be used if they help illustrate the process and are easily recognized as being non-interactive



- Scroll bars appear if the container sizes would otherwise cause the dialog to be larger than 440 pixels wide by 320 pixels high.
- The wizard does not check values that are entered by the users— it is up to your batch script to do this.

## Wizard page

Each wizard page is set up by WizardPage Block, defined between a 'WizardPage begin' and 'WizardPage end' statement. You need as many WizardPage blocks as there are pages in the wizard. Each of the areas 'contains' different things and is therefore known as a 'container'. The size and contents of each container must be specified. An example wizard block is:

```
WizardPage begin "WizardName"#title for first page
  title "Page1Name" #if a new wizard
  container begin "Image"
    container information           #usually image info
    ...                               #on first page
  container end
  container begin "PageControls"  #control buttons
    ask action "< Back"
    ask action "Next >" goto label1
    ask action "Cancel" close goto label2
  container end
  wizard close
WizardPage end
```

## Container

A container block defines the size, contents and layout of a single 'container' or 'pane' in a wizard page. Any number of containers can be defined though realistically there will be only two or three per page. For example,

```
container begin "con2"
  container height_pct 20
  container below "con1"
    ask action "< Back"
    ask action "Next >" goto wizard_page2
    ask action "Cancel" close goto wizard_page2
  container end
```

## Navigation buttons container

The last container on any page should contain the standard navigation buttons.

```
container begin "con2"
  container height_pct 12
  container below "con1"
    ask action "< Back"
    ask action "Next >" goto wizard_page2
    ask action "Finish" goto check_parameters_and_close
    ask action "Cancel" close goto wizard_close
  container end
```

The buttons should be programmed to work in the following way:

- **<Back:** Returns to the previous page. It should be greyed out on the first page. (This is achieved by an **ask action** command with no specified **GoTo Label**; as per the "Back" button in the example above). When the user presses the Back button they should be presented with any choices they have made up to that point rather than the original defaults. Therefore defaults should be set up before the first Wizard Block.
- **Next>:** Moves to the next page in the sequence, maintaining whatever settings the user provides in previous pages. This is greyed out on the last page.
- **Finish:** Applies the settings defined in the wizard (either default or input by the user) to ER Mapper and completes the task. The finish button must appear on the last page and should be included at any point that the wizard can complete the task (even the front page if there are reasonable defaults).

**Cancel:** Discards any settings specified in the wizard, terminates the process, and closes the wizard window. The Cancel button is always the right most button.

A single page wizard has only a Finish and a Cancel button.

For a consistent look multi-page wizards should have Next and Back buttons on all pages. However, the Next button should be replaced by a Finish button on the last page, and the Back button should be greyed out on the first page.

## Ask and Show commands

All the "Ask", "Say" and "Title" commands, such as

```
Ask Yesno "Show Contours:"
$contours_yn
```

work within a Wizard block. However, the 'Ok', 'Cancel' and 'Help' buttons which are automatically added to Ask Blocks do not appear in Wizard blocks.

In addition, a number of Ask commands are designed especially for specifying input into wizards. These are documented in "*Dialog box input fields*" below. They include, for example, 'Ask Action' which draws the navigation buttons on the bottom of the wizard page.

### **Show Image "Filename"**

Shows the image TIF file in the container, following the same rules that other Ask/Show items do. The image is shown at its true size, not fitted to the container size. The image files default directory is 'icons'.

### **General guidelines for wizard pages**

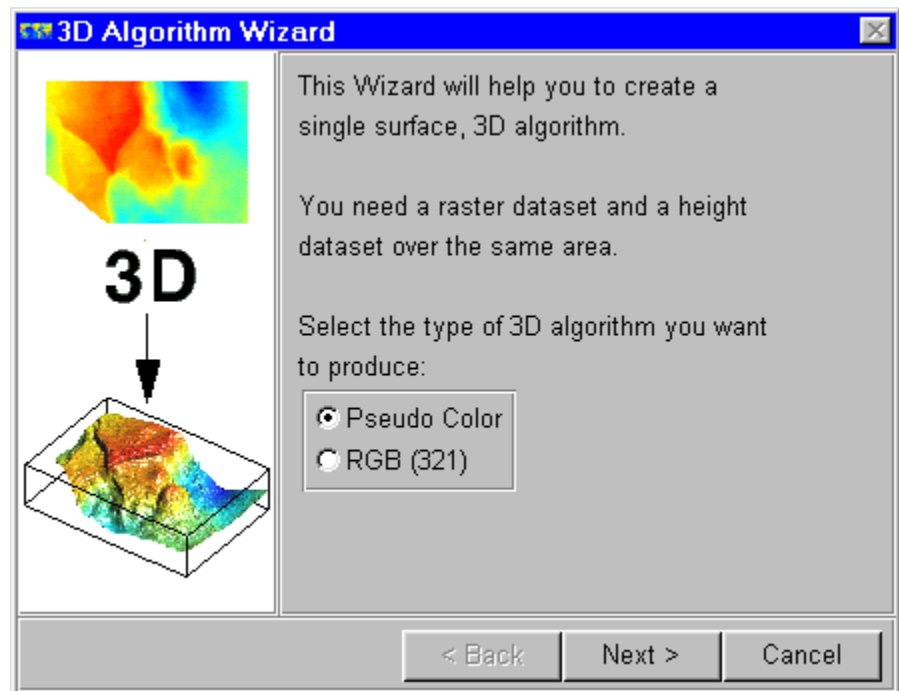
- Wizards should have Windows XP look and feel.
- Pages should flow linearly through a series of steps. It is possible to jump from one wizard to another but this should be avoided as it will confuse the user. The user should not have to use any functions outside of the wizard to complete a task.
- Wizard pages should be easy to understand without having to read them very carefully. It is better to use more simple pages than fewer complex pages.
- If it isn't obvious, the last page should give the user information about how to proceed when the wizard is finished.
- Wizards don't automatically progress to the next page. the user may have neglected to provide all necessary information, in which case a 'Wizard Error' will be reported. The wizard does not proceed until the appropriate field is filled in.
- Use a conversational writing style, with words like you' and 'your', contractions and short, common words.
- Ask users what they would like to do rather than telling them what to do. Thus, use 'which option do you want...' or 'would you like...' instead of 'choose a layout'.
- Avoid using technical terminology that may be confusing to a novice user.
- Use as few words as possible.
- Keep the writing clear, concise and simple, but don't be condescending.
- It is quite possible (as with ask forms) to decide under program control if containers or ask/say commands are to be added to a wizard form. This makes it quite powerful.
- If a label name of a 'labels\_left' container item is too long, part of it will not be displayed. If this occurs, either change to 'labels\_above' or shorten the label.

- It is not possible to have wizard dialogs which close automatically when, for example, a process is completed. The containers must have a control (**Save, Close, Finish** etc.) that explicitly closes the dialog.

### Example wizard

Below is an example of a wizard script showing the created wizard pages. This and other examples can be found in the 'batch' directory.

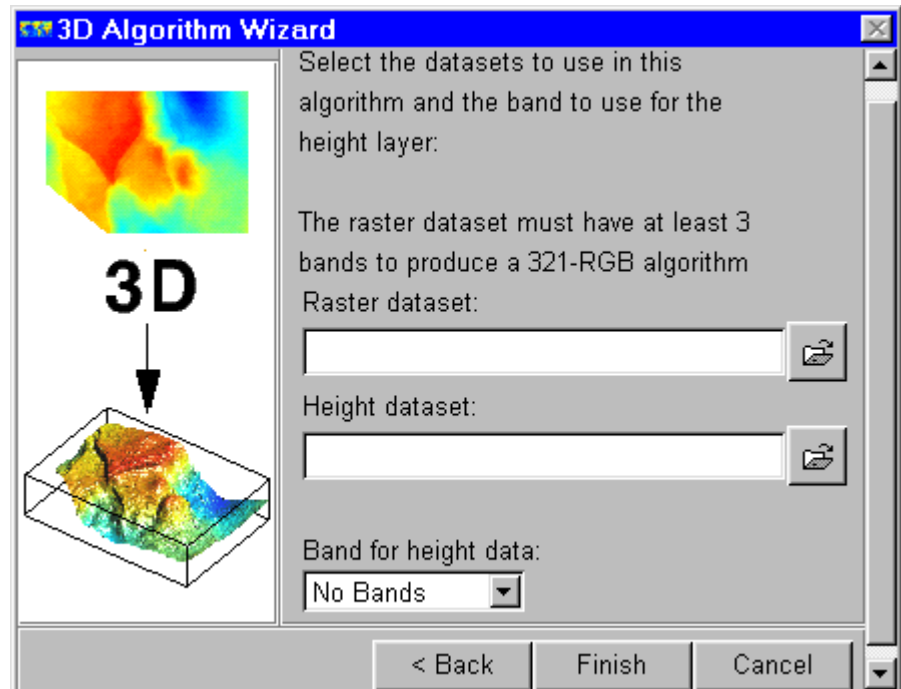
```
#
# Script: 3D_Wizard.erb
#
# Summary: Wizard to create a standard single surface
3D algorithm
#
# Details: Ask for a pseudo or RGB alg, choose the
datasets,
#         create an algorithm, copy it to the window and
#         set the view mode to 3D. Simple!
#
include "lib/BE_Startup.erb"
$yesno_Pseudocolor = "1"
$yesno_RGB = "0"
$listmenu["0"] = "Psuedo Color"
$listmenu["1"] = "RGB"
$list_choice = $listmenu["0"]
$RasterDataset = ""
$HeightDataset = ""
$lut = "greyscale"
#####
#####
```



```

wizard_page_1:
Wizard begin "3D Algorithm Wizard"
  title "3D Algorithm Wizard"
  container begin "Mode"
    container items labels_left
    container right "Image"
    say "This Wizard will help you to create a"
    say "single surface, 3D algorithm."
    say ""
    say "You need a raster dataset and a height"
    say "dataset over the same area."
    say ""
    say "Select the type of 3D algorithm you want"
    say "to produce:"
    ask listmenu_exclusive "" $listmenu $list_choice
  container end
  container begin "PageControls"
    container height_pct 12
    container below "Mode"
    container items horizontal right justify
    ask action "< Back"
    ask action "Next >" goto CheckPseudo_or_RGB
    ask action "Cancel" close goto WizardCancel
  container end
  container begin "Image"
    container width_pixels 131
    container height_pixels 280
    container left "Mode"
    container above "PageControls"
    show image "standard_icons/Wizards/3D_wiz"
  container end
Wizard end
CheckPseudo_or_RGB:
  if ($list_choice == "RGB") then goto
wizard_page_2_rgb
  goto wizard_page_2_pseudo
#####
#####

```

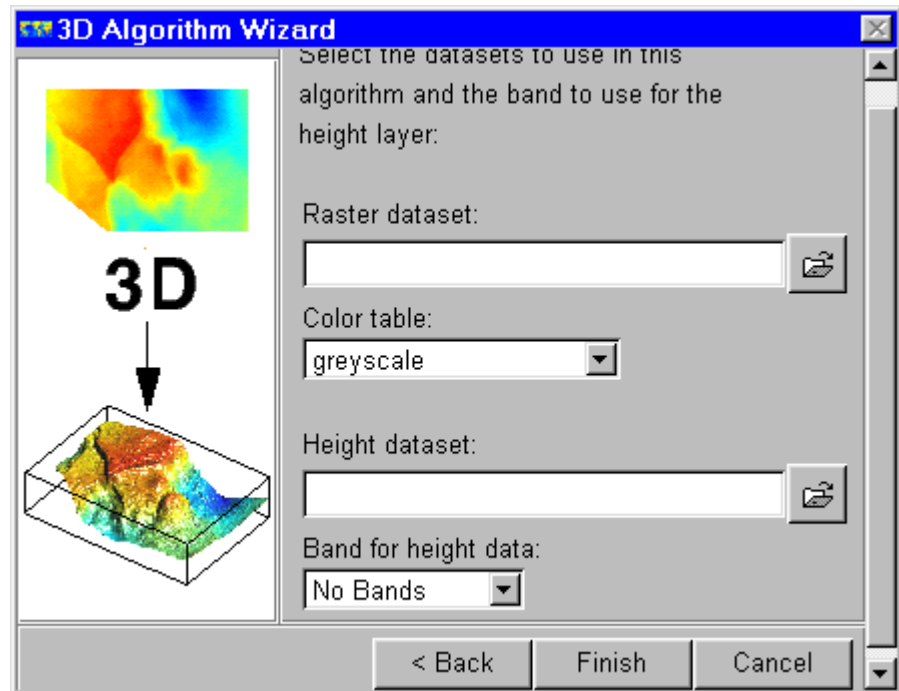


```
wizard_page_2_rgb:
Wizard begin "3D Algorithm Wizard"
  title "3D Algorithm Wizard"
  container begin "InputDatasets"
    container items labels_above
    container right "Image2"
    say "Select the datasets to use in this"
    say "algorithm and the band to use for the"
    say "height layer:"
    say ""
    say "The raster dataset must have at least 3"
    say "bands to produce a 321-RGB algorithm"
    ask file "Raster dataset: " ".ers"
  $RasterDataset
    ask file "Height dataset: " ".ers" $HeightDataset
    say ""
    ask bandmenu "Band for height data:"
  $HeightDataset
  $bandmenu
  container end
  container begin "Image2"
    container width_pixels 131
    container height_pixels 280
    container above "PageControls2"
    show image "standard_icons/Wizards/3D_wiz"
  container end
```

```

container begin "PageControls2"
  container height_pct 12
  container below "InputDatasets"
  container items horizontal right justify
  ask action "< Back" goto wizard_page_1
  ask action "Finish" goto WizardFinish
  ask action "Cancel" close goto WizardCancel
container end
Wizard end
#####
#####

```



```

wizard_page_2_pseudo:
Wizard begin "3D Algorithm Wizard"
  title "3D Algorithm Wizard"
  container begin "InputDatasets"
    container items labels_above
    container right "Image2"
    say "Select the datasets to use in this"
    say "algorithm and the band to use for the"
    say "height layer:"
    say ""
    ask file "Raster dataset: " ".ers"
  $RasterDataset
    ask lutmenu "Color table:" $lut
    say ""
    ask file "Height dataset: " ".ers" $HeightDataset
    ask bandmenu "Band for height data:"
  $HeightDataset $bandmenu
  container end

```

```

container begin "Image2"
    container width_pixels 131
    container height_pixels 280
    container above "PageControls2"
    show image "standard_icons/Wizards/3D_wiz"
container end
container begin "PageControls2"
    container height_pct 12
    container below "InputDatasets"
    container items horizontal right justify
    ask action "< Back" goto wizard_page_1
    ask action "Finish" goto WizardFinish
    ask action "Cancel" close goto WizardCancel
container end
Wizard end
#####
#####
Warn1:
    say warning "You must choose a Raster dataset"
    if ($list_choice == "RGB") then goto
wizard_page_2_rgb
    goto wizard_page_2_pseudo
Warn2:
    say warning "You must choose a Height dataset"
    if ($list_choice == "RGB") then goto
wizard_page_2_rgb
    goto wizard_page_2_pseudo
WizardFinish:
    if ($RasterDataset == "") then goto Warn1
    if ($HeightDataset == "") then goto Warn2

wizard close
    if ($list_choice == "RGB") then goto DoRGB
    goto DoPseudo
#####
#####
DoPseudo:
    $alg = new algorithm
    set $alg description "Pseudo Color 3D"
    set $alg mode pseudo
    set $alg lut to $lut
    $layer1 = first layer
    set $layer1 description "Raster"
    set $layer1 dataset to $RasterDataset
    $lasttran= last transform
    set $lasttran limits to actual
    add height layer
    $layer2 = current layer
    set $layer2 description "Height"
    set $layer2 dataset to $HeightDataset
    set $alg view mode to perspective

```



```

new window
copy algorithm to window
go window
goto FinishUp
#####
#####
DoRGB:
  $alg = new algorithm
  set $alg description "RGB 3D"
  set $alg mode rgb
  $layer1 = first layer
  set $layer1 type red
  set $layer1 description "Red"
  set $layer1 dataset to $RasterDataset
  set $layer1 input 1 to band 1
  $lasttran= last transform
  set $lasttran limits to actual
  add green layer
  $layer2 = current layer
  set $layer2 description "Green"
  set $layer2 dataset to $RasterDataset
  set $layer2 input 1 to band 2
  $lasttran= last transform
  set $lasttran limits to actual
  add blue layer
  $layer3 = current layer
  set $layer3 description "Green"
  set $layer3 dataset to $RasterDataset
  set $layer3 input 1 to band 3
  $lasttran= last transform
  set $lasttran limits to actual
  add height layer
  $layer4 = current layer
  set $layer4 description "Height"
  set $layer4 dataset to $HeightDataset
  set $alg view mode to perspective
  new window
  copy algorithm to window
  go window
  goto FinishUp
#####
#####
WizardCancel:
FinishUp:
exit

```

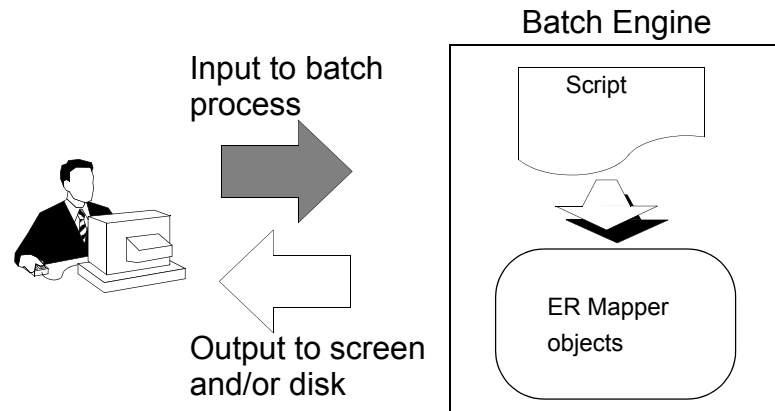


# Scripting language

Batch scripts usually perform the following overall functions:

- Input data or arguments from a user
- Perform actions on specified objects
- Output the results of the actions to screen and/or to disk

## Input to batch process.



Once a batch script has been invoked by a command from the user, it generally requires the input of data or arguments before it can perform any processing. In some unusual situations this input information is "hard coded" into the script so that, apart from editing the script, the user has no control over the batch process. This is generally sufficient for "one-off" scripts which are developed to perform a very specific set of tasks. It is often preferable to have inbuilt default arguments which are used by the batch process in the absence of any from the user. This makes the batch script more flexible and, thus, more efficient.



*You can also pass arguments to scripts from menu (.erm), and toolbar (.bar) files and dynamic link choosers. See the ["Menu and toolbar files \(.erm\) and \(.bar\)"](#), and ["Dynamic links menu"](#) chapters.*

- Methods for the user to input data are as follows:
- Include arguments in the command line string.
- Provide on-screen dialog boxes with controls and fields
- Provide wizards which interactively prompt users to input data.

## Command line arguments

This method is used when the user is running the batch file from a command line using the `ermapper -b` command. The command string syntax is as follows:

```
ermapper -b "batch_script [argument1 argument2]"
```

For example:

```
ermapper -b "copy_file image1.ers image2.ers"
```



*All arguments after 'ermapper -b' need to be enclosed in double quotes (" ").*

The hypothetical batch script, `copy_file.erb` copies a image file **image1.ers**, specified by `argument1`, to file **image2.ers**, specified by `argument2`.

To run your script from a command line you must have the following line at the end of the script:

```
exit n
```

Where *n* is the exit code returned from ER Mapper.

- *n* = 0: exits the script and leaves ER Mapper open.
- *n* = >0: exits the script and closes ER Mapper.

## Dialog boxes

There are a number of commands which create dialog boxes for users to enter data.

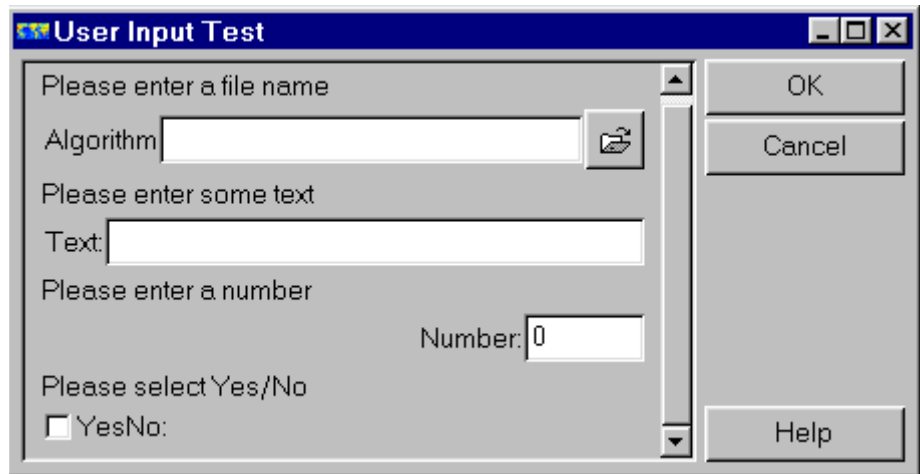
The following script brings up a dialog box for input into ER Mapper. The default title is "batch engine input". The `ask begin/end` block allows multiple inputs to be obtained from the user. The `say` statements create a single line of text in the dialog box. The `ask` statements create an alphanumeric entry box. The example below opens a dialog box with the title 'User Input Test' and asks for an algorithm file name, some text, a number and a yes-or-no answer.

```
ask begin
  title "User Input Test"
  say "Please enter a file name"
  ask file "Algorithm:" ".alg" $file_name_input
  say "Please enter some text"
  ask text "Text:" $some_text_input
  say "Please enter a number"
  ask number "Number:" $some_number_input
  say "Please select Yes/No"
  ask yesno "YesNo:" $yesno_input
ask end
```

- The input dialog is opened when the "ask end" statement is read.
- There are a number of different types of field.

- The input variables can be arrays.
- To get multiple lines of text use multiple ask statements.

The above example will create the dialog box shown below:



### Dialog box input fields

You can create the following types of input fields in dialog boxes Refer to [“Script commands - alphabetical listing”](#) below for descriptions of the command.

Type	Description	Script command
Band chooser	Entry box with a button to open a chooser dialog listing the band descriptions for the specified file. You can also specify multiple or single choice.	<b>ask bandchooser</b>
Band menu	Drop down selection list of descriptions for all the bands in a specified file.	<b>ask bandmenu</b>
Color chooser	Alphanumeric entry box with a button to open a color chooser dialog	<b>ask colorchooser</b>
Datum chooser	Alphanumeric entry box with a chooser button to open the datum chooser dialog	<b>ask datum</b>
<b>Directory chooser</b>	Alphanumeric entry box with a chooser button for entering a directory name.	<b>ask directory</b>
Exclusive generic list	List of defined entries with exclusive radio buttons.	<b>ask listmenu_exclusive</b>
File	Alphanumeric entry box with a file chooser button for entering a file name.	<b>ask file</b>

Type	Description	Script command
Generic list	Selection list containing defined entries	<b>ask listmenu</b>
<b>Grid layer</b>	Drop down selection list of descriptions for all the gridding layers in a specified project file.	<b>ask gridlayermenu</b>
Hardcopy driver	Alphanumeric entry box with chooser button to open either the Windows printer driver or the ER Mapper hardcopy driver selection dialog box.	<b>ask hardcopy</b>
Link	Alphanumeric entry box with a link chooser button for entering a dynamic link name	<b>ask link</b>
Lookup table	Drop down selection list of the Color Lookup tables for selection.	<b>ask lutmenu</b>
<b>Navigation buttons</b>	Button to select a defined action	<b>ask action</b>
Projection chooser	Alphanumeric entry box with a chooser button to open the projection chooser dialog.	<b>ask projection</b>
Text and number	Alphanumeric entry box	<b>ask text number</b>
Yes/No	Check box for selecting an option.	<b>ask yes/no</b>

## Wizards

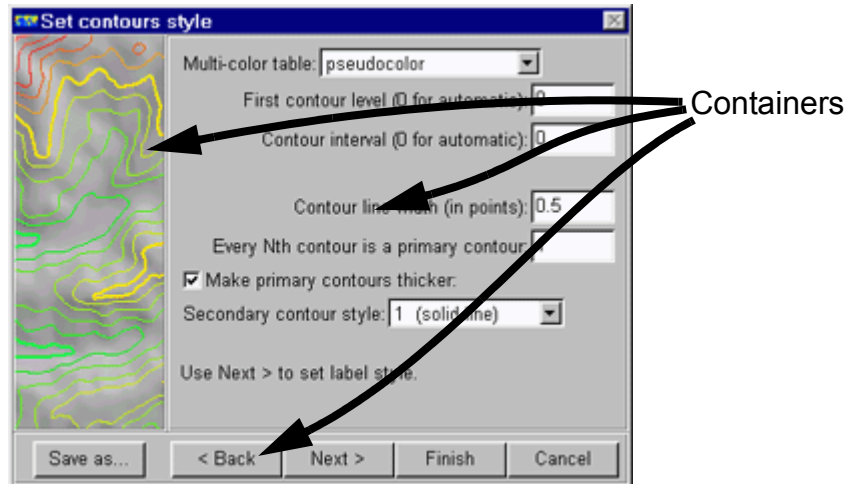
Wizards consist of a sequence of dialogs or pages through which a user is led. Each wizard page contains a number of containers. These wizard page containers have the same input fields as normal dialog boxes. In the scripting language, wizard pages and containers are defined by `WizardPage begin..... WizardPage end`, and `container begin.... container end` statements.

```

WizardPage begin "Wizard Title
.....
  container begin
    .....
    container end
  container begin
    .....
    container end
.....
WizardPage end

```

An example wizard dialog page is shown below:



The following commands are used to create wizards. Refer to ["Script Commands - alphabetical listing"](#) below for descriptions of the commands

Function	Description	Script command
<b>Wizard Page block</b>	Starts a new wizard page	<b>WizardPage begin end</b>
<b>Close wizard</b>	Closes the wizard	<b>Wizard close</b>
<b>Container block</b>	Creates a container within a wizard page.	<b>container begin end</b>
<b>Container button justification</b>	Justifies button position	<b>Container right left justify</b>
<b>Container position</b>	Specifies how this container is to be placed in relation to another container	<b>Container above below left right</b>
<b>Container item positions</b>	Specifies the direction of placement of the items in a container	<b>Container Items</b>
<b>Container label positions</b>	Specifies where the labels for an item in the container will appear	<b>Container Labels</b>
<b>Container size</b>	Specifies the container width and height as a percentage of the remaining space.	<b>Container width height</b>
<b>Display image</b>	Displays image in container	<b>show image</b>

### Using preferences to remember settings

ER Mapper maintains a list of preferences that can be retrieved at any time. Using this facility, you can design wizards to "remember" values entered by users so that they do not have to re-enter them when they run the wizard again.

For example: a wizard could ask the user to enter a background color which defaults to "white". This value could be stored in a variable `$background_color`. The value in `$background_color` could then be stored as a preference. When the wizard is run again, it could retrieve the `$background_color` value from the preference and thus avoid having the user enter it again to change it from the default value of "white".

### To set a preference

In the above example you use the following command to set the preference:

```
set preference "Wizard:Image:BackgroundColor"
$background_color
```

This command assigns the value in variable `$background_color` to a preference named "Wizard:Image:BackgroundColor".

You can use any preference name, but it should be meaningful and non-ambiguous. ER Mapper wizards use the "Class:Name:Variable" format made up as follows:

- Class: (e.g. Wizard)
- Name: (e.g. Image)
- Variable (e.g. BackGroundColor)

### To retrieve a preference

In the above example you use the following command to retrieve the preference:

```
$background_color = get preference
"Wizard:Image:BackgroundColor" "white"
```

This command assigns the value in preference "Wizard:Image:BackgroundColor" to variable `$background_color`. It defaults to "white" if the preference has not been set.

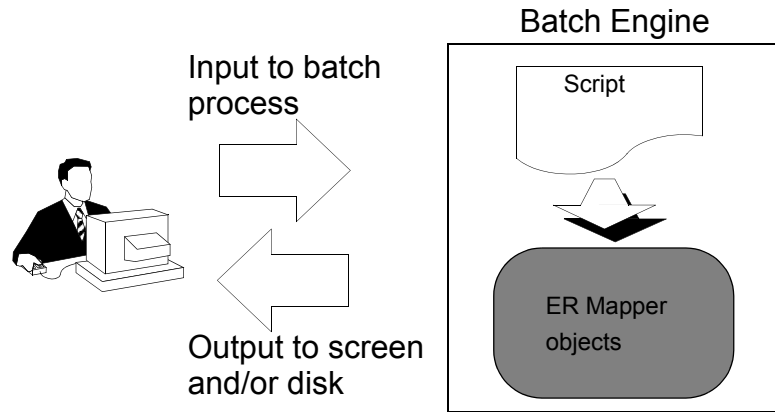


---

*For syntax information see the ["Preferences"](#) and ["Script Commands - alphabetical listing"](#) sections below.*

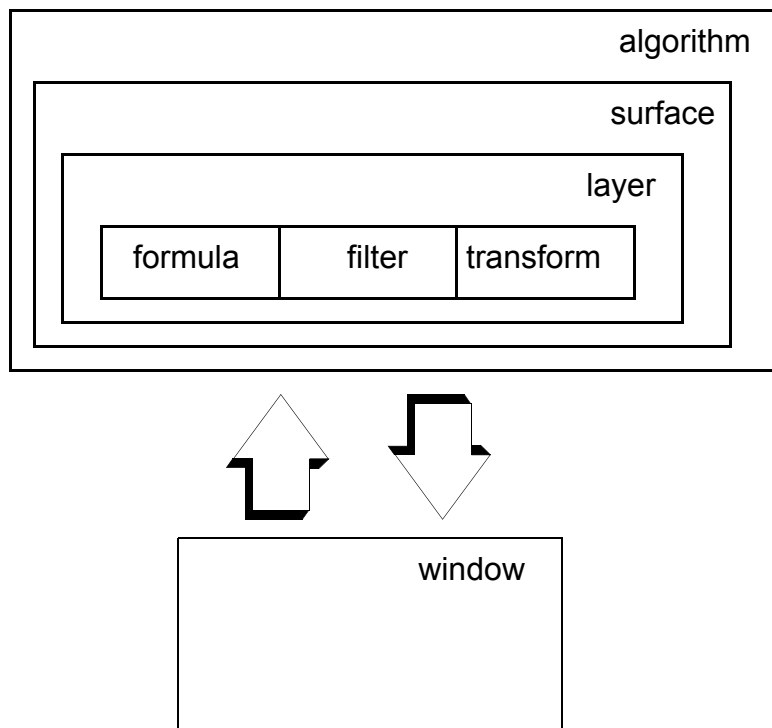


# ER Mapper Objects.



ER Mapper comprises a number of objects which, in turn, contain attributes. Batch scripts instruct the batch engine to perform actions on these objects and their attributes. These actions can include creating new instances of an object or setting specific attributes pertaining to those objects.

The diagram below illustrates the objects and their relationships to one another:



An **algorithm** object will contain **surface** objects which, in turn, contain **layer** objects. The **layer** objects contain **formula**, **filter** and **transform** objects.

The **Window** object is used for displaying algorithms copied to it.

All the objects can exist on their own within the batch engine; e.g. you can have a defined layer object that is not contained within a surface or algorithm. It is also possible to have an empty window. However, you can only copy an algorithm (with its contained objects) to a window to display it.

You can assign an object to a variable (e.g. an algorithm can be assigned to `$alg`), and then use this variable to define that object in your script. You can also set the batch engine to point to a specific object; i.e. make it current. Any batch commands that do not name a specific object will be performed on the current object.

For example,

```
select $alg[1]#makes $alg[1]the current algorithm
set algorithm mode to rgb
```

has the same effect as

```
set $alg[1] mode to rgb
```

## Image Manipulation

When you write scripts to manipulate ER Mapper images and algorithms, the algorithms you construct and edit with the scripting language are within the batch engine—quite separate to ER Mapper itself.

Thus, if you create an algorithm using a script, to view it you need to copy it to ER Mapper.

Similarly, say you have an algorithm already being viewed and edited in ER Mapper. A script to change the Color mode from Pseudocolor to HSI must copy the algorithm from ER Mapper into the batch engine, make the appropriate color mode change, and then copy the resulting algorithm back to ER Mapper.

In the batch engine there are a number of reserved words which point to objects.

- `current window`: indicates the current window. When a batch script starts, the current window is set to point to the currently active window in the GUI.
- `current algorithm`: indicates the current algorithm within the batch engine
- `current surface`: indicates the current surface within the algorithm
- `current layer`: indicates the current layer within the surface
- `current formula`: indicates the current formula within the layer
- `current transform`: indicates the current transform within the layer
- `current filter`: indicates the current filter within the layer
- `current input`: indicates the current layer input within the layer

window, algorithm, surface, layer, transform, filter, and input usually refer to the current object. Some examples:

```
$win1 = current window
```

means set \$win1 to the current window.

```
copy window
```

means copy the current window, and update the current window pointer to refer to the new window.

```
$win = copy window
```

means copy the current window, update the current window to refer to the new window, and set \$win to refer to the new window.

```
select $win
```

means make the window defined by the variable \$win the current window.

To give you a feel for how ER Mapper and the batch engine interact with each other here are some examples to work through. These are all from the ERMAPPER\batch directory. Please look through the other script files for more extensive examples.

This first example copies the algorithm to the batch engine, the included file sets up the layers for the colordrape algorithm, then the lut and algorithm description are set. The changed algorithm must be copied back to the window before the user sees the result.

```
#from Create_CD.erb to create a colordrape algorithm
copy algorithm from window

#include code to create the colordrape layers
include "lib/Create_CD.erb"

set algorithm lut to "pseudocolor"
set algorithm description to "Colordrape"

copy algorithm to window
exit
```

This more extensive example is part of lib\Clip\_99\_All\_Active\_Layers.erb. It is included in the 'Go\_Limits\_99.erb' script. The algorithm has already been copied to the batch engine.

```
# Run the Algorithm at 100x100 resolution
#
go algorithm 100 100

# Cycle through all layers setting limits to actual data limits
#
first active raster layer
if ($ERROR != 0) then goto no_algorithm
next_active_layer:

last transform
set transform limits to actual
next active raster layer
if ($ERROR == 0) then goto next_active_layer

# Run the Algorithm at 100x100 resolution.
#
go algorithm 100 100

# Cycle through all layers setting the transform clip to 99.0%
#
first active raster layer
if ($ERROR != 0) then goto no_algorithm

next_active_layer:
last transform
set transform clip to 99.0
next active raster layer
if ($ERROR == 0) then goto next_active_layer

no_algorithm:
```

## Actions

The batch engine controls the objects by performing actions on them. The following table lists these actions and shows their applicable objects. [See "Script Commands - alphabetical listing"](#) below for a full description of each command. The specific page numbers for the commands are shown in brackets

Command	Object types							Comment
	windo w	algo- rithm	surface	laye r	trans -form	form- ula	filter	
add			*	*	*	*	*	Add object to a containing object.
copy	*	*	*	*	*	*	*	Duplicate object but do not insert into another object.
copy to copy from	*							Copy algorithm to and from window for display
current	*	*	*	*	*	*	*	Point to current object
delete	*	*	*	*	*	*	*	Delete object
duplicate			*	*	*	*	*	Make a duplicate of the object and insert it into the containing object
first, last, next, previous	*	*	*	*	*	*	*	Point to object and make it current
fit page		*						Fit algorithm page to hardcopy device.
get		*	*	*	*	*	*	Get the value of the specified attribute.  See <a href="#">"Attributes"</a> below for a list of the attributes
go	*	*						Run the object
go background	*							Run object in background

Command	Object types							Comment
	windo w	algo- rithm	surface	laye r	trans -form	form- ula	filter	
load		*		*		*	*	Load algorithm, formula or filter into the batch engine.
move			*	*				Move to new position.
new	*	*	*	*	*	*	*	Create new object
save		*				*	*	Save object to file.
select	*	*	*	*	*	*	*	Make specified object current
set	*	*	*	*	*	*	*	Set the value of the specified attribute. See <a href="#">"Attributes"</a> below for a list of the relevant attributes.
turn on turn off			*	*				Turn object on or off

## Attributes

The objects have attributes associated with them. You can use the **set** and **get** commands to either set the attributes to required values or interrogate the object to return the values of specific attributes. The following table lists the attributes and shows which objects they are associated with. It also indicates whether you can perform sets and/or gets on them. See *"Script Commands - alphabetical listing"* for a full description of each command.

Attribute	Object types						Comment
	algorithm	surface	laye r	trans -form	formul a	filter	
azimuth			set/get				Sun shade azimuth
background	set						Specifies the background color as an rgb value or a color defined by a color variable.
cell sizex cell sizey			get				Cell size
cell type			get				Cell type

Attribute	Object types						Comment
	algorithm	surface	layer	transform	formula	filter	
clip				set			Percentage clip
color			set/get				Link layer color specified by rgb values or defined in a colorval variable.
contents	set/get						Page contents extents
contents extents	set						Set page contents extents to algorithm extents.
coordsys	set/get						Sets the coordinate system type.
dataset			set/get				Layer image file name
datum	set/get						Sets the datum
description	set/get	set/get	set/get			set/get	Description text
edit program			set/get				Edit program name (dlink)
editable			set/get				Editable flag true or false
elevation			set/get				Sun shade elevation on or off
equalize				set			Set to histogram or gaussian equalize
formula			get/set/load/save		load/save		Sets the formula as a text value, or loads a formula (.frm) file.
init program			set/get				Init program name
input count			get				Number of inputs
input filter count			get				Number of filter inputs
input to band					set		Formula input to band
input transform count			get				Number of input transforms
input output				set/get			Input/output limits
layer count	get	get					Number of layers
limits				set			Transform limits
link extension			set/get				Link file extension
link type			set				Sets link type to mono- or truecolor

Attribute	Object types						Comment
	algorithm	surface	layer	transform	formula	filter	
lut	set/get	set/get					Sets the color table for the specified surface or first surface in an algorithm.
matrix						set/get	Element in filter matrix
mode	set/get	set/get					Sets the color mode for the specified surface or first surface in an algorithm.
mosaic type	set/get						Sets the mosaic type to overlay or feather
name		set					Name text
output filter count			get				Number of output filters
output transform count			get				Number of output transforms
page autovary_value	set						Calculate autovary value
page border	set/get						Page borders
page center	set						Center image on page; yes/no
page constraints	set/get						Page constraints
page extents	set						Page extents
page scale	set/get						Page scale
page size	set/get						Standard page size, e.g "US Letter"
page topleft page bottomright	set/get						Page extents coordinates
page width page height	set/get						Inside dimensions of page.
page view mode	set/get						View image only or with page layout.
params						set/get	Parameters
postsampled						set/get	Postsampled process flag
projection	set/get						Image projection
rotation	set/get						Image rotation



Attribute	Object types						Comment
	algorithm	surface	layer	transform	formula	filter	
rows cols						set/get	Number of rows and columns.
scale						set/get	Filter scale
shading			set/get				Sets sunshading to on or off.
supersample type	set						Supersample type
threshold						set/get	Filter threshold value
opleft bottomright	set/get						Sets the extents.
transparency		set					Surface transparency
type			set/get	set/get		set/get	Object type
units	set/get						Measurement units
userfile						set/get	Source file name
userfunc						set/get	Function name
view mode	set/get						View mode (2D/3D)
zoffset		set					Surface Z offset
zscale		set					Surface Z scale

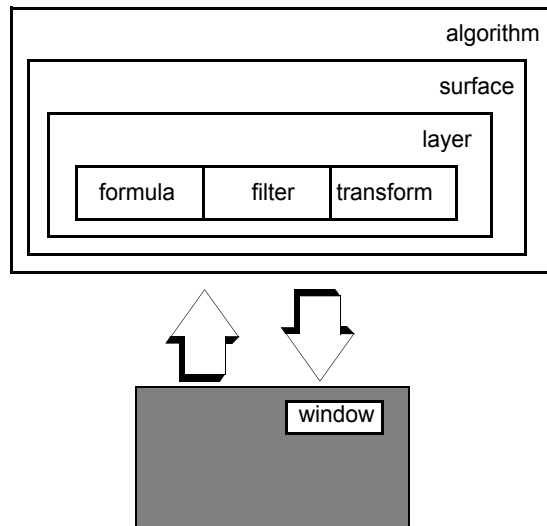
### Command summaries

The following sections summarize the commands applicable to the different objects.



Refer to ["Script Commands - alphabetical listing"](#) for more information on the commands.

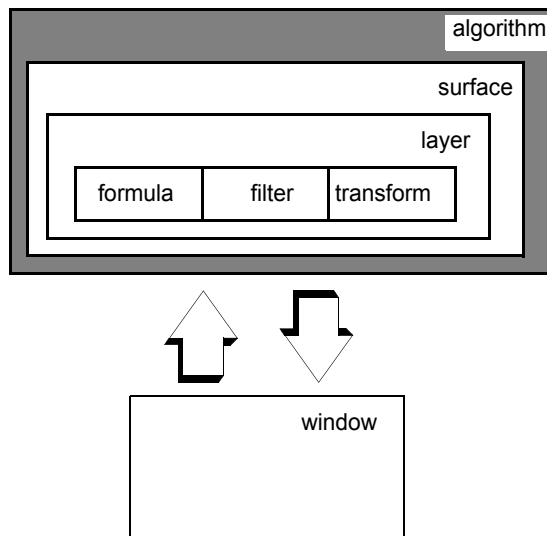
## Windows



Description	Command
Copies the window reference from win1 to win2	<b>\$win2 = \$win1</b>
Copies the algorithm from the current or specified window to the batch engine and set the current algorithm pointer to point to it.	<b>copy algorithm from window \$win</b>
Copies the specified algorithm or that indicated by the current algorithm pointer to the current window in ER Mapper.	<b>copy algorithm to window</b>
Makes a duplicate of the current or specified window and its algorithm	<b>copy window \$win</b>
Sets pointer to the current window.	<b>current window</b>
Deletes the current or specified window and its algorithm.	<b>delete window \$win</b>
Updates the current window pointer to point to the oldest ,newest, next oldest or next newest window open.	<b>first last next previous window</b>
Runs the algorithm in the current or specified window as a background task	<b>go background window \$win</b>
Runs the algorithm in the current or specified window.	<b>go window \$win</b>
Opens a new image window, with a default algorithm	<b>new window [x y w h]</b>
Open the designated dialog box on screen	<b>open window</b>

Description	Command
Zoom in or out using specified option	<b>previouszoom</b> <b>zoom in out</b> <b>zoom to</b>
Updates the current window pointer to point to the given window.	<b>select \$win</b>
Sets the mouse pointer to the specified mode: zoom, zoombox, roam(hand) or pointer.	<b>set pointer mode to</b> <b>zoom zoombox roam pointer</b>
Various geolink functionality	<b>set window geolink mode</b>

## Algorithms

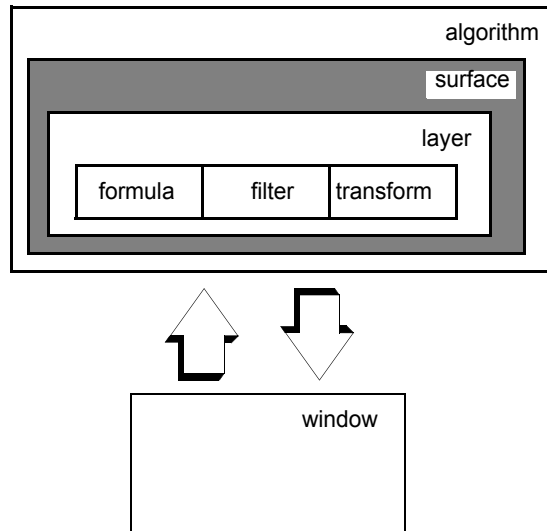


Description	Command
Sets \$alg2 to refer to \$alg1	<b>\$alg2 = \$alg1</b>
Copies the current or specified algorithm.	<b>copy algorithm</b>
Sets the pointer to the current algorithm	<b>current algorithm</b>
Deletes all references to an algorithm.	<b>delete algorithm</b>
Changes the current algorithm pointer to point to the first, last, next or previous algorithm.	<b>first last next previous algorithm</b>
Gets the current or specified algorithm's description.	<b>get algorithm description</b>

Description	Command
Gets the algorithm's coordinate system type.	<b>get algorithm coordsys</b>
Gets the algorithm's datum or projection type.	<b>get algorithm datum projection</b>
Gets the number of layers in the current or specified algorithm.	<b>get algorithm layer count</b>
Gets the algorithm's pseudocolor LUT name.	<b>get algorithm lut</b>
Gets the algorithm's mode.	<b>get algorithm mode</b>
Gets the algorithm's mosaic type.	<b>get algorithm mosaic type</b>
Gets the algorithm's extents.	<b>get algorithm topleft bottomright eastings  longitude meters_x</b>
Gets the algorithm's units or rotation. Rotation is in decimal degrees, units is a units string.	<b>get algorithm units rotation</b>
Runs the current or specified algorithm.	<b>go algorithm [width height][match]</b>
Loads the given algorithm.	<b>load algorithm \$name</b>
Creates a new (empty) algorithm.	<b>new algorithm</b>
Saves the current or specified algorithm with the given name.	<b>save algorithm \$name</b>
Saves the current or specified algorithm as a virtual dataset	<b>save algorithm as virtual dataset \$name</b>
Saves the current or specified algorithm as a dataset	<b>save algorithm as dataset</b>
Changes the current algorithm pointer to point to the specified algorithm.	<b>select \$alg</b>
Changes the algorithm's background color to the given RGB values or to a value specified by a variable.	<b>set algorithm background to \$red \$green \$blue</b> <b>set [algorithm] background to colorval \$colorvariable</b>
Sets the algorithm's coordinate system type to the given type.	<b>set algorithm coordsys to \$csys raw en ll</b>
Sets the algorithm's datum or projection to the given type. This will cause layers of an incompatible type to be turned off within the algorithm.	<b>set algorithm datum projection</b>
Changes the current or specified algorithm description to the given text.	<b>set algorithm description to \$text</b>
Changes the pseudocolor LUT for the first surface in the algorithm to the given lut file.	<b>set algorithm lut [to] \$lutname</b>

Description	Command
Changes the processing mode for the first surface in the algorithm to the given mode.	<b>set algorithm mode to \$mode pseudo rgb hsi</b>
Sets the mosaic type (where different datasets overlay) to the given type	<b>set algorithm mosaic type</b>
Sets the current or specified algorithm supersample type.	<b>set algorithm supersample type</b>
Sets the current or specified algorithm topleft and bottomright extent fields in easting/northing, latitude/longitude or raw coordinate systems.	<b>set algorithm topleft bottomright</b>
Sets the units or rotation to the given value.	<b>set algorithm rotation units to \$units</b>

## Surfaces

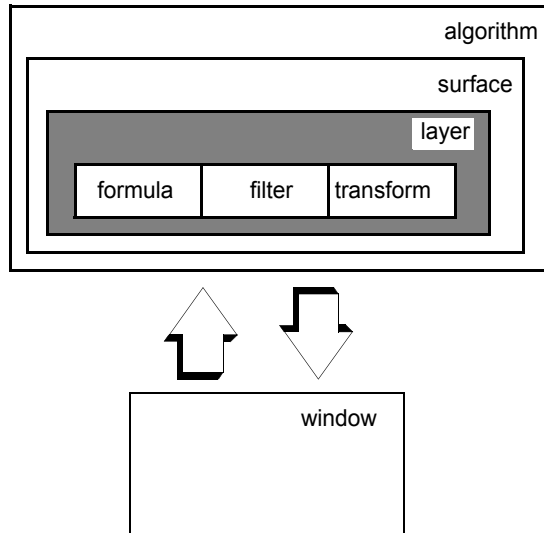


If an algorithm has more than one surface, they have to be individually defined and configured by these commands. This is not necessary if the algorithm has only one surface because the algorithm commands default to the top surface.

Description	Command
Adds the specified surface to the current algorithm.	<b>add \$srf</b>
Copies a the current or specified surface, but does not add the new surface to any algorithm	<b>copy surface</b>

Description	Command
Sets the pointer to the current surface.	<b>current surface</b>
Deletes the current or specified surface from the current algorithm.	<b>delete surface</b>
Duplicates the current or specified surface within the current algorithm.	<b>duplicate surface</b>
Selects a surface within the current algorithm and makes it current.	<b>first last previous next surface</b>
Gets the current or specified surface description.	<b>get surface description</b>
Gets the number of layers in the current or specified surface.	<b>get surface layer count</b>
Moves the current surface within the current algorithm.	<b>move surface up down top bottom</b>
Creates a new surface but does not add it to any algorithm.	<b>new surface</b>
Selects the given surface and makes it the current surface.	<b>select \$srf</b>
Changes the current or specified surface description to the given text.	<b>set surface description to \$text</b>
Sets the Color Table for the current surface.	<b>set surface lut</b>
Sets the color mode for the current surface.	<b>set surface mode</b>
Sets the current surface name to the given name.	<b>set surface name</b>
Sets the Z Scale, Z Offset or transparency of the current surface to \$value.	<b>set surface zscale zoffset transparency</b>
Checks whether current surface is active, and returns TRUE (1) for active and FALSE (0) for inactive.	surface active
Enables/Disables the current surface.	<b>turn surface on off</b>

## Layers



All layer commands are related to the current surface. For example, an **add layer** command will add the layer to the current surface. If there is no designated current surface then it will default to the top surface within the current algorithm.

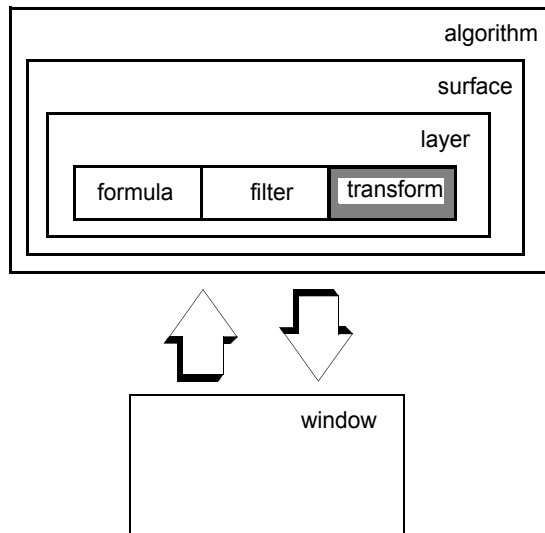
Description	Command
Assigns specified layer (e.g. \$lay1) to another variable (e.g. \$lay2)	<b>\$lay2 = \$lay1</b>
Adds the current or specified layer to the current surface, at the end of the layer list.	<b>add layer</b>
Adds a new layer of the given type after the current layer in the current surface.	<b>add layer</b>
Copies the current or specified layer but does not insert it into a surface.	<b>copy layer</b>
Sets the pointer to the current layer.	<b>current layer</b>
Deletes the current or specified layer. When all layers in a surface are deleted, the surface will also be deleted unless it is the only surface on the algorithm.	<b>delete layer</b>
Duplicates the current layer in the current surface.	<b>duplicate layer</b>
Sets the current layer pointer to point to the specified layer in the current surface.	<b>first last next previous layer</b>
Gets the current or specified layer color. Valid only on link layers.	<b>get [layer] color</b>

Description	Command
Gets the layers sun shading azimuth/elevation.	<b>get layer azimuth elevation</b>
Gets the number of bands in the current or specified layer.	<b>get layer band count</b>
Gets the nominated band description of the current or specified layer.	<b>get layer band description</b>
Reads the current or specified layer cell size.	<b>get layer cell sizex sizey</b>
Gets the current or specified layer's image cell type.	<b>get layer cell type</b>
Gets the EN or LL coordinates from the given cellX and cellY values.	<b>get layer x_coordinate y_coordinate from \$cellX \$cellY</b>
Gets the current or specified layer's image name.	<b>get layer dataset</b>
Gets the current or specified layer's description.	<b>get layer description</b>
Gets the current or specified layer's editable flag. Valid only on link layers.	<b>get layer editable</b>
Gets the current or specified layer's edit init program name.	<b>get layer edit init program</b>
Gets a copy of the current or specified layer's formula.	<b>get layer formula</b>
Gets the number of inputs in the current or specified layer.	<b>get layer input count</b>
Gets the number of input filters in the current or specified layer input.	<b>get layer input filter count</b>
Gets the number of input transforms in the current or specified layer input.	<b>get layer input transform count</b>
Gets the link file extension string for the current or specified layer.	<b>get layer link extension</b>
Gets the number of output filters in the current or specified layer.	<b>get layer output filter count</b>
Gets the number of output transforms in the current or specified layer.	<b>get layer output transform count</b>
Gets the current or specified layer's shading value.	<b>get layer shading</b>
Gets the current or specified layer's type.	<b>get layer type</b>
Checks whether the current layer is active, and returns TRUE (1) for active and FALSE (0) for inactive.	layer active



Description	Command
Loads the formula file into the current or specified layer's formula.	<b>load layer formula</b>
Moves the current or specified layer within the current surface to the given position within its surface.	<b>move layer up down top bottom</b>
Creates a new layer of the given type.	<b>new layer</b>
Moves the pointer to the next layer of the type specified.	next [active][raster vector] layer (type)
Saves the current or specified layer's formula to the formula file.	<b>save layer formula</b>
Sets the current layer pointer to point to the specified layer.	<b>select \$lay</b>
Sets the sun shade azimuth/elevation to the given value in degrees.	<b>set layer azimuth elevation</b>
Sets the current or specified layer color to the given RGB or color values. Valid only on link layers.	<b>set layer color</b>
Sets the image file name for the current or specified layer to \$dsname.	<b>set layer dataset</b>
Changes the layer description to the text given.	<b>set layer description</b>
Sets the editable flag for current or specified layer. Valid only for link layers.	<b>set layer editable [to] true false</b>
Sets the current or specified layer's edit/init program to the given name. Valid only on link layers.	<b>set layer edit init program</b>
Sets the current or specified layer formula.	<b>set layer formula</b>
Sets the designated layer input to the specified band number	<b>set layer input \$n1 to band \$n2</b>
Sets the link file extension of the current or specified layer to the given string.	<b>set layer link extension</b>
Sets the link type of the current or specified layer to monocolour or truecolour. Valid only on link layers.	<b>set layer link type to monocolour truecolour</b>
Turns sunshading on/off for the current or specified layer.	<b>set layer shading on off</b>
Changes the current or specified layer type to the given type.	<b>set layer type</b>
Turns the current or specified layer on or off.	<b>turn layer on off</b>

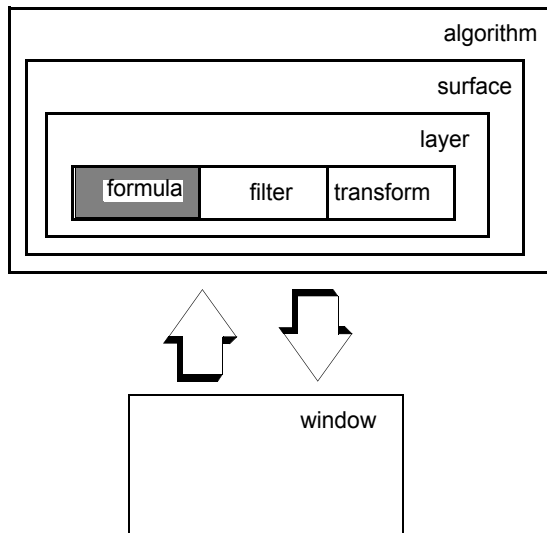
## Transforms



Description	Command
Assign specified transform (e.g. \$tra1) to another variable (e.g. \$tra2).	<b>\$tra2 = \$tra1</b>
Adds the specified transform after the current transform in the current layer.	<b>add \$tra</b>
Adds the specified transform after the current transform in the given stream input of the current layer.	<b>add \$tra to layer input</b>
Adds a transform of the given type after the current transform.	<b>add transform</b>
Adds a point to the current or specified transform at x, y.	<b>add transform point</b>
Makes a duplicate of the current or specified transform but does not insert it into the current algorithm.	<b>copy transform</b>
Sets the pointer to the current transform.	<b>current transform</b>
Deletes the transform, and all references to it.	<b>delete transform</b>
Copies the current transform in the current layer, and inserts the copy after the current transform.	<b>duplicate transform</b>
Sets the current transform to the first, last, next or previous transform in the current layer. Optionally select an input transform, and/or a transform type.	<b>first last next previous transform</b>

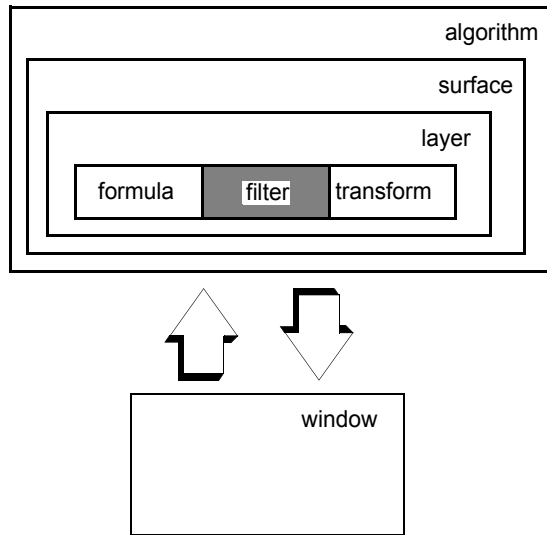
Description	Command
Gets the current or specified transform's limits.	<b>get transform input output min max</b>
Gets the current or specified transform type.	<b>get transform type</b>
Matches the existing output transforms of all layers to the current or specified layer in the same surface.	<b>match transform [to \$layer]</b>
Creates a new transform which then becomes the current transform.	<b>new transform</b>
Sets the current transform to the specified transform.	<b>select \$tra</b>
Applies a clip of \$pct percent to the current or specified transform.	<b>set transform clip</b>
Sets the current or specified transform input/output limits.	<b>set transform input output min max</b>
Sets the current or specified transform limits to \$percent or actual.	<b>set transform limits to actual \$percent</b>
Sets the current or specified transform's output limits to the input limits.	<b>set transform output limits to input limits</b>
Applies a gaussian equalize operation to the current or specified transform.	<b>set transform to gaussian equalize</b>
Applies a histogram equalize operation to the current or specified transform.	<b>set transform to histogram equalize</b>
Sets the current or specified transform type.	<b>set transform type</b>

## Formula



Description	Command
Adds the current or specified formula to the current layer.	<b>add formula</b>
Makes a copy of the current or specified formula, but does not add it to a layer.	<b>copy formula</b>
Sets the pointer to the current formula.	<b>current formula</b>
Deletes the current formula in the current layer.	<b>delete formula</b>
Duplicates the current formula and adds it to the current layer.	<b>duplicate formula</b>
Sets the current formula pointer to point to the specified formula in the current layer.	<b>first last next previous formula</b>
Loads the current or specified formula file.	<b>load formula \$fname</b>
Creates a new formula.	<b>new formula</b>
Saves the current or specified formula to the formula file.	<b>save formula to \$fname</b>
Sets the current formula to the specified formula.	<b>select \$for</b>
Sets the current or specified formula (\$formula is a string).	<b>set formula to \$formula</b>
Sets the current formula input to the image band given.	<b>set input \$n1 to band \$n2</b>

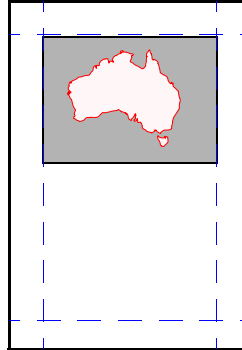
## Filters



Description	Command
Assigns specified filter (e.g. \$fil1) to new variable (e.g. \$fil2)	<b>\$fil2 = \$fil1</b>
Adds the current or specified filter to the current layer after the current filter.	<b>add filter</b>
Makes a copy of the current or specified filter but does not insert the new filter into a layer.	<b>copy filter</b>
Sets the pointer to the current filter.	<b>current filter</b>
Deletes the filter and all references to it.	<b>delete filter</b>
Copies the current filter, and inserts the copy after the current filter.	<b>duplicate filter</b>
Sets the current filter to the first, last, next or previous filter in the current layer.	<b>first last next previous filter</b>
Gets the current or specified filter description.	<b>get filter description</b>
Gets an element from the current or specified filter matrix.	<b>get filter matrix</b>
Gets the current or specified user filter's parameters.	<b>get filter params</b>
Gets the current or specified filter's post sampled process flag.	<b>get filter postsampled</b>
Gets the number of rows/columns in the current or specified filter.	<b>get filter rows cols</b>

Description	Command
Gets the scale or threshold for the current or specified filter. Valid only on convolution and threshold filters.	<b>get filter scale threshold</b>
Gets the current or specified filter's type.	<b>get filter type</b>
Gets the current or specified user filter source file name.	<b>get filter userfile</b>
Gets the current or specified user filter function name. Valid only on usercode filters.	<b>get filter userfunc</b>
Loads the current or specified filter from the given file.	<b>load filter</b>
Creates a new filter, but does not add it to any layer.	<b>new filter</b>
Saves the current or specified filter to the given file.	<b>save filter</b>
Sets the current filter to the specified filter.	<b>select \$fil</b>
Changes the current or specified filter description to the given text.	<b>set filter description to \$text</b>
Sets an element of the current or specified filter matrix. Valid only on convolution and threshold filters.	<b>set filter matrix</b>
Set the current or specified user filter parameter string. Valid only on usercode filters.	<b>set filter params to \$paramstring</b>
Sets whether the current or specified filter can process resampled data, or source data.	<b>set filter postsampled</b>
Sets the number or rows/columns for the current or specified filter to \$number.	<b>set filter rows cols</b>
Sets the scale or threshold for the current or specified filter. Valid only on convolution and threshold filters.	<b>set filter scale threshold</b>
Sets the filter type of the current or specified filter.	<b>set filter type</b>
Sets the current or specified user filter source file, for a C usercode filter. Valid only on usercode filters.	<b>set filter userfile [to] \$filename</b>
Sets the current or specified user filter function name. Valid only on usercode filters.	<b>set filter userfunc to \$funcname</b>

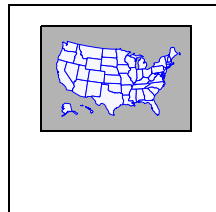
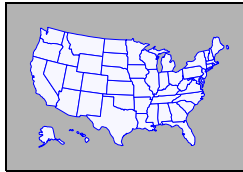
## Page Setup



Description	Command
Sets the page width and height of the current or specified algorithm to that of the currently specified hardcopy device.	<b>fit page to hardcopy</b>
Gets the page contents extents coordinates.	<b>get contents topleft bottomright</b>
Gets the page borders of the current or specified algorithm.	<b>get page top bottom left right border</b>
Get the page constraints of the current or specified algorithm.	<b>get page constraints</b>
Gets the page scale of the current or specified algorithm.	<b>get page scale</b>
Gets the page size of the current or specified algorithm.	<b>get page size</b>
Gets the page extents coordinates.	<b>get page topleft bottomright</b>
Gets the specified page parameter of the the current or specified algorithm.	<b>get page width height</b>
Calculates and sets the bottom right contents extents coordinate, based on the topleft coordinate, page size, borders, and scale.	<b>set contents bottomright from topleft</b>
Sets the page contents extents coordinates to that specified by \$value.	<b>set contents topleft bottomright</b>
Calculates one of the page setup variables based on the Constraints setting.	<b>set page autovary_value</b>
Centers the contents of the page horizontally or vertically.	<b>set page center horizontal vertical</b>
Sets the page constraints of the current or specified algorithm.	<b>set page constraints to \$constr zoom page border scale</b>

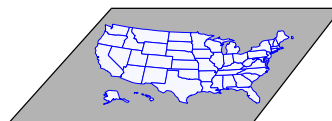
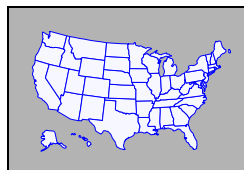
Description	Command
Calculates and sets the page extents of the algorithm based on the contents extents, the borders, and the scale.	<b>set page extents from contents</b>
Sets the page scale to a specific number or the maximum scale possible.	<b>set page scale to \$scale max</b>
Sets the page size of the current or specified algorithm.	<b>set page size to \$size</b>
Sets the page extents coordinates to that specified in the variable \$value.	<b>set page topleft bottomright</b>
Sets the page borders.	<b>set page top bottom left right border</b>
Sets the specified page size parameter of the current or specified algorithm.	<b>set page width height</b>
Set the contents extents to the algorithm extents (where <b>algorithm</b> is the current algorithm).	<b>set contents extents to [algorithm] extents</b>

### Page view mode



Description	Command
Gets the page view mode of the current or specified algorithm.	<b>get page view mode</b>
Sets the page view mode of the current or specified algorithm to normal or layout.	<b>set page view mode to \$pvmode normal layout</b>

### View mode





Description	Command
Gets the view mode of the current or specified algorithm.	<b>get view mode</b>
Sets the view mode of the current or specified algorithm to 2D, 3D perspective or 3D flythru.	<b>set view mode to \$vmode 2d perspective flythru</b>

## Preferences

Preferences enable batch scrips to retain parameters when they are shut down and re-run. There are a number of preferences already defined in ER Mapper, but you can define and set new ones.

The following commands write to the user's preference file.



Refer to "[Script Commands - alphabetical listing](#)" for descriptions of these commands.

Function	Description	Script command
<b>Set preference entry</b>	Adds or updates the named preference entry	set preference
<b>Get preference entry</b>	Returns the value in the named preference entry. Can return a default value if the entry does not exist.	get preference
<b>Set color preference entry</b>	Adds or updates the named color preference entry	set preference
<b>Get color preference entry</b>	Returns the value in the named color preference entry. Can return a default value if the entry does not exist.	get preference

## File and Path separators

Function	Description	Script command
<b>Build an absolute file specification</b>	Builds the absolute file specification from a given relative file specification and its parent directory	build absolute filespec <parent_dir> <rel_file>
Build a file path name	Builds a complete path for a file from up to 4 given elements.	build file path <element_1> <element_2> ....<element 4>
<b>Build a relative file specification</b>	Builds file specification relative to its parent directory from a given absolute file specification.	build relative filespec <parent_dir> <abs_file>

Function	Description	Script command
Convert file separators	Converts the file separators in a given file specification from English to native and vice versa.	convert <filespec> to english native sep separator
<b>Get English file separator</b>	Returns the standard English file separator used by the host workstation or PC. This would be "\" on a Win32 (PC) platform or "/" on a Unix platform.	get english file sep separator
<b>Get native path file separator</b>	Returns the path separator used natively by the host workstation or PC.	get native path file sep separator

## Other commands

There are a number of commands that control the batch processing environment. These are listed below:

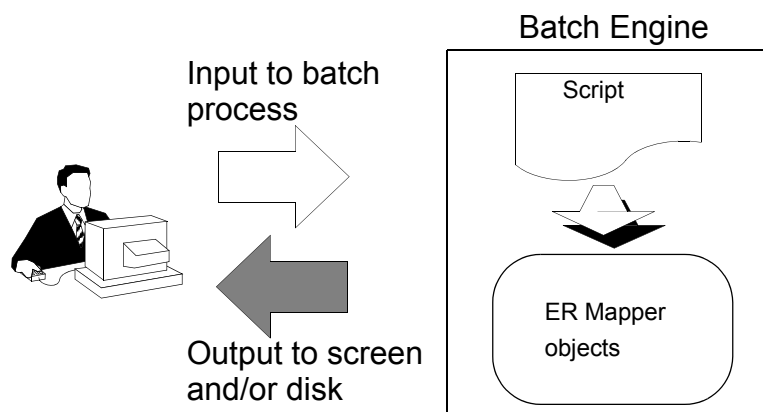


Refer to ["Script Commands - alphabetical listing"](#) for descriptions of these commands.

Function	Description	Script command
<b>get file size</b>	Gets the given file's size in bytes.	get \$filename size
<b>get free space on disk</b>	Gets the amount of free space, in bytes, on the given disk.	get \$diskname free space
<b>get environment variable</b>	Gets the given environment variable.	getenv \$envname
<b>set environment variable</b>	Sets the given environment variable.	setenv \$envname
<b>delete directory or file</b>	Deletes a directory or file. If the file name has an ".ers" extension, the raster file is also deleted.	delete \$dir \$file
<b>list contents of directory</b>	Lists the contents of the specified directory into an array.	listdir
<b>edit text file</b>	Edits the given file name in a text editor. The file is created if it doesn't exist.	edit \$filename
exit batch process	Exits the batch process and returns you to the ER Mapper main window. See also <a href="#">"Command line arguments"</a> .	exit

Function	Description	Script command
file exists	Verifies the existence of the specified file and returns true or false.	if \$filename exists
<b>split string into array</b>	Splits the given string into an array of substrings at the given token	split \$string at \$token
<b>format number of digits after decimal point</b>	Formats a number to have a specified number of digits after the decimal point .	format \$input_value \$precision
<b>get ER Mapper version number</b>	Returns the ER Mapper version number as a string or as a number.	get version_string version_number
<b>execute system command</b>	Executes a system command	system \$command
<b>system command status dialog</b>	Similar to the system syntax, except that a progress dialog is created.	system \$command status ["title"]
<b>File name, extension and path keywords</b>	Returns the file name, path and extension of a given file specification	<b>filename \$string</b> <b>dirname \$string</b> <b>fileext \$string</b>
<b>Color keywords</b>	You can use the <b>color_red</b> , <b>color_green</b> and <b>color_blue</b> keywords to get the rgb (0-255) components of a color variable.	<b>\$bg_color color_red</b> <b>\$bg_color color_green</b> <b>\$bg_color color_blue</b>

## Output.



The results of a batch process can be directed to an image window, to a Batch Engine Dialog box, or to be saved as a file.

## Output to image window

To view an algorithm on the screen, you have to copy it to a window object. The following commands load an algorithm into the current algorithm object, and then copy it to the current window.

```
load algorithm "test_algorithm.alg"  
copy algorithm to window
```

## Output to file

You can copy instances of the following object types to a file using the 'save' command:

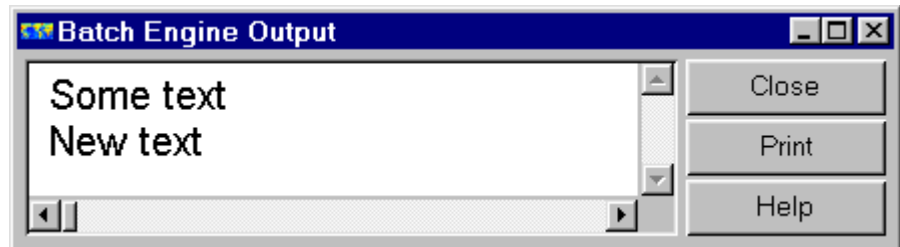
- algorithm
- formula
- filter

Examples of the save commands are given below:

```
save algorithm "my_alg.alg"  
save algorithm as dataset "my_image.ers"  
save algorithm as virtual dataset "my_image_vds.ers"  
save layer formula "my_form.frm"  
save formula "my_form.frm"  
save filter "my_filter.ker"
```

## Output to Batch Engine Output dialog (print commands)

The Batch Engine Output dialog opens automatically when you use 'print' commands. New print commands append text to that what exists in already open dialogs.



There are two print commands:



---

*print no CR/LF*  
*println CR/LF*

By default, numbers are printed to 6 decimal places. Use the ~ to specify a different number of decimals.

statement	result
<code>print 1</code>	1.000000
<code>\$var = 2</code> <code>print \$var</code>	2.000000

statement	result
<code>\$text = "Hello World"</code> <code>print \$text</code>	Hello World
<code>\$text = "Hello World"</code> <code>print \$text</code> <code>println \$text</code>	Hello WorldHello World
<code>print ~3 1</code>	1.000

The print and println commands write to an output dialog by default. Alternatively, you can print out to a file using:

```
set output to $filename
```

The print or println commands in a batch script after this set command will create the file if it doesn't already exist and write the output to the end of the file.

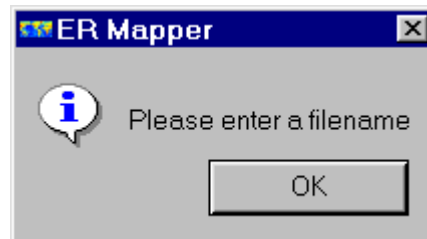
To reset the output to write to a dialog use the following command:

```
set output to output window.
```

## Warning dialog

You can use the 'say warning' command to open an ER Mapper warning dialog box with a specified message. The following is an example of its use:>

```
checkdataset:
if ($filename!= "") then goto DatasetOK
say warning "Please enter a filename"
goto wizard_page_1
DatasetOK:
```



## Status dialog

You can create a status dialog box which indicates the progress, as a percentage and/or as text, of any process the user has invoked. This is usually used in wizards

```
...
ask action "Status" goto OpenStatus
...
OpenStatus:
open status
$percent = 20
say status $percent "Opening image window\n"
....
goto wizard_page_1
```



The following commands create and write to a status or warning dialog box.



*There is only one status dialog. If it is already open, new messages will be added to it.*



Refer to [“Script Commands - alphabetical listing”](#) for descriptions of the commands.

Script command	Description
open status	Opens status dialog
say status	Adds message to status dialog
<b>delete status</b>	Deletes status dialog
<b>say warning</b>	Opens warning dialog with a message.

## Library of batch scripts

Common batch script functions are stored in %ERMAPPER%\batch\lib. These can be included in your scripts. For example, if you include lib\BE\_Startup.erb, you get \$machine\_type, \$ERMAPPER, \$ERMBIN and \$ERMSCRIPTS and a number of other variables defined for you.

Browse through the directory to see what is available. Some of the scripts include others, so study them to see how to include the code in your script.

# Scripting reference

This chapter lists and describes all the operators, functions, variables, keywords and commands used in the ER Mapper batch scripting language.



---

*All variable names, label, and keywords are case insensitive.*

## Operators

The full suite of standard operators is available.

- #: Specifies a comment line. For example,

```
# this is a comment
```

- = \* / + - %: Standard arithmetic operators: assignment, multiplication, division, addition, subtraction, modulus
- != <> > >= < <= ==: Standard comparison operators: not equal to, not equal to, greater than, greater than or equal to, less than, less than or equal to, equal to

- ~: Specifies the number of decimal places to print for numbers. For example,

```
print ~3 $var  
prints $var to 3 decimal places.
```

- (): For assigning operator precedence.
- []: Array specifier.

## Concatenation

The Batch engine can add numbers to strings. Numbers are converted into text and joined to the end of strings.

### Example

```
$number = 6  
$string = "datasetname"  
$string = $string + $number + ".ers"
```

## Mathematical functions

- $\sin(x)$ ,  $\cos(x)$ ,  $\tan(x)$ : The sine, cosine and tangent of the angle  $x$ , with the angle specified in radians.
- $\text{asin}(x)$ ,  $\text{acos}(x)$ ,  $\text{atan}(x)$ : The arcsine, arccosine and arctangent of the angle  $x$ , with the angle specified in radians.
- $\text{pow}(x,y)$ : The number  $x$  raised to the power of  $y$ .
- $\log(x)$ ,  $\text{exp}(x)$ : The natural log and exponent of  $x$ .

- `sqrt(x)`: The square root of `x`.
- `floor(x)`: The number `x` rounded down.
- `ceil(x)`: The number `x` rounded up.
- `min(x,y)`, `max(x,y)`: The minimum and maximum of two numbers `x` and `y`.
- `abs(x)`: The absolute value of `x`.

## Variables

Variables have a `$` leading character, followed by a letter and then alphanumeric characters (including underscores).

Syntax: `$variablename`

Memory is allocated dynamically as assignments are made.

```
$var = 1
$var2 = $var
```

ER Mapper supports global name spaces for variables; i.e values set in included scripts affect parent scripts.

Memory is freed as variables are deleted or when the script exits.

There are a number of types. Once a variable has been defined, its type becomes fixed. For example, in the first statement below, the assignment `$a = 0` defines the variable `$a` as a number variable. The second statement `$a = "Hello"` tries to assign a string to the number variable and produces an error.

```
$a = 0
$a = "Hello"
```

The following types of variable are available.

- **Number:** A floating point number. Numerical variables can use the full suite of arithmetic operators: `-` `*` `/` `+` `-` `%`.

For example,

```
$var = 1 + 1
```

is equivalent to

```
$var = 2
$var2 = $var * 100
```

is equivalent to

```
$var2 = 200
```

- **String:** Strings are enclosed in quotes. Allowable arithmetic operators are: `+` `+=`

```
$hello = "Hello "
$world = "World"
$hello_world = $hello + $world
```

gives the result



"Hello World"

- **YesNo:** Allowable values are:
  - yes or 1
  - no or 0
- **Mode:** Allowable values are:
  - pseudocolor (or pseudo)
  - rgb
  - hsi
- **LayerType:** Allowable values are:
  - pseudocolor
  - red
  - green
  - blue
  - hue
  - saturation (or sat)
  - intensity (or int)
  - classification (or class)
  - classdisplay
  - link
  - height
- **CoordSys:** Allowable values are:
  - raw
  - en
  - ll
- **MosaicType:** Allowable values are:
  - overlay
  - feather

- **PageViewMode:** Allowable values are:
  - normal
  - layout
- **TransformType:** Allowable values are:
  - linear
  - exp
  - log
  - hist
- **ViewMode:** Allowable values are:
  - 2d
  - perspective
  - flythu
- **FilterType:** Allowable values are:
  - convolution
  - threshold
  - user
- **SuperSampleType:** Allowable values are:
  - nearest
  - bilinear
- **References:** You can also have string references to a:
  - Window
  - Algorithm
  - Layer
  - Formula
  - Transform
  - Filter
- **Color:** A three part number variable which can be specified in two ways:

- As a set of three numbers which represent the red, green and blue components of the color. The syntax is:

```
$Color = [colorval] r,g,b
```

For example,

```
$Color = 124,5,5
```

```
$Color = colorval 124,5,5
```

- As a string containing one of the named colors listed in the ER Mapper color chooser. The syntax is:

```
$Color = colorval "known_color"
```

For example,

```
$Color = colorval "red"
```

- **CellType:** Allowable values are:
  - uint8
  - uint16
  - uint32
  - int8
  - int16
  - int32
  - ieee4
  - ieee8
- **MachineType:** Can be used for creating system command paths, etc. Allowable values are:
  - sun4
  - sun5
  - irix5
  - decalpha
  - hp
  - win32

## Arrays

You can have arrays of any type.

```
$array[1]
```

The array index can be a variable: `$array[$var]`. For example,

```
$count = 3
$filename[$count] = "vegetation"
```

Multi-dimensional arrays are supported. For example,

```
$array[cars][red] = 1
```



*You can specify the array index to start at any number, including a negative value. We do, however, recommend that you restrict this to 0 or 1.*

## Page size options

The page size options listed in the page setup dialog are special strings that are recognised by ER Mapper. They can be included in a listmenu chooser. For example,

```
$p_size_array[0] = "Custom"
$p_size_array[1] = "US Letter"
$p_size_array[2] = "US Legal"
$p_size_array[3] = "A1"
$p_size_array[4] = "A2"
$p_size_array[5] = "A3"
ask listmenu "Choose a page size from the list" "Page
size chooser" $p_size_array $p_size_choice
```

## Keywords

ER Mapper supports the following keywords

above	absolute	action	active
actual	add	algorithm	all
annotation	as	ask	at
autovary_value	azimuth	background	band
bandchooser	bandmenu	batch	batch
begin	below	blue	border
bottom	build	cell	center
centre	class	classdisplay	classification
clip	close	color	color_blue
color_green	color_red	colorchooser	colorval
colour	colour_blue	colour_green	colour_red
colourchooser	colourval	cols	columns
constraints	container	contents	coord
coordinate	copy	count	current
dataset	datasets	datum	defaults
delete	description	dirname	down

duplicate	edit	editable	elevation
else	end	equalise	equalize
exists	exit	extension	extention
extents	false	file	fileext
filename	filter	first	fit
format	formula	free	from
gaussian	geolink	geoposition	get
getenv	go	goto	green
hardcopy	height	horizontal	hsi
hue	if	image	in
include	info	init	int
intensity	items	job	justify
last	layer	left	limits
link	listmenu	listmenu_exclusive	load
lut	lutmenu	main	match
matrix	mode	monocolor	monocolour
mosaic	move	name	new
newline	next	none	number
off	on	open	out
overview	page	parameters	params
path	point	pointer	postsampled
preference	preferences	previous	print
println	process	profile	program
projection	pseudo	pseudocolor	raster
realtime3d	red	relative	rgb
right	roam	roam	rows
sarturation	sat	save	say
scale	scattergram	screen	select
sep	separator	set	setenv
setup	shading	show	size
sizeX	sizeY	sleep	space
split	status	sunangle	supersample
surface	system	tempname	text
then	threshold_value	timestamp	title

to	top	transform	transparency
traverse	true	truecolor	truecolour
turn	type	up	userfile
userfunc	vector	version_number	version_string
vertical	view	virtual	warning
width	width_pct	width_pixels	window
wizard	wizardpage	yesno	zoffset

## Flow control

This section documents the flow control commands, with examples.

### Labels

For labelling the program for `goto`s. (A `goto` always refers to a label within the same file; i.e local name space. This means that ER Mapper can distinguish between a label in the main code and the same one in an included file. Labels can be used for conditional or unconditional control.

- **Syntax:** labelname:

For example,

```
get_filenames:
```

### Controls

- **goto:** Unconditional control. For example,

```
label1:
goto label1
```

- **if ... then ... else:** Conditional control. For example,

```
label1:
$var = 1
if ($var == 1) then goto label1
if ($var == 2) then goto label1 else goto label2
```

### Explicit exit

```
if ($var == 1) then goto carryon
exit
carryon:
```




---

*If you are running the batch script from the command line using **ermapper -b**, the script must exit with a return code > 0 (e.g. `exit 1`) to ensure that the ER Mapper application also exits. If the script exits with `exit` or `exit 0`, then the command line prompt will "hang" until you physically stop the ER Mapper application.*

## Looping

```
$count = 0
increment:
$count = $count + 1
if ($count <= 10) goto increment
```

## Including files

Files must be specified by their absolute path or a path relative to the ERMAPPER\batch directory. Please note that ER Mapper supports forward (/) slashes for PC and Unix directory paths. It supports backward slashes (\) on PCs only. To maintain portability between platforms it is advisable to use forward (/) slashes.

- Syntax: include "filename"

Example:

```
include "lib/BE_Startup.erb"
```

## Error reporting

Some commands return an error code and text which are stored in the following variables:

`$ERROR:` stores the error code

- 0 - successful
- 1 - unsuccessful
- `$ERROR_TEXT:` stores the error text message

Commands that return an error code include: previous, next, first, last, load, save

## Script Commands - Alphabetical listing

This section lists all the script commands alphabetically, and describes their operation and syntax.

---

### absolute|relative path

Specifies whether the variable returned by 'ask file' or 'ask link' will be an absolute filepath or relative to the current directory. The default is absolute.

**Syntax:** absolute|relative path

#### Example

```
relative path
ask file "File to contour:" ""$dsname ".ers,.alg" $dsname
```

---

## add (new) layer

Adds a new layer of the given type after the current layer in the current surface.

**Syntax:** `add $stype|pseudocolor|red|green|blue|hue|saturation|intensity|height|classification|classdisplay|link layer`

**\$stype:** Layer type variable: see [“Variables”](#) for allowed values.

### Examples

```
add pseudocolor layer
```

This is the same as

```
new pseudocolor layer
add layer
```

---

## add (new) transform (to layer)

Adds a transform of the given type after the current transform. If no type is specified, it defaults to linear.

**Syntax:** `add [$ttype|linear|exp|log|hist] transform`

**\$ttype:** Transform type variable: value can be linear, exp, log or hist

### Examples

```
add transform
add linear transform
```

---

## add filter (to layer)

Adds the current or specified filter to the current layer after the current filter.

**Syntax:** `add [$ftype|convolution|threshold|user] filter|$fil`

**\$ftype:** Filter type variable: value can be convolution, threshold or user.

### Examples

```
add $fil
add user filter
```

---

## add formula (to layer)

Adds the current or specified formula to the current layer.

**Syntax:** `add formula|$for`

### Example

```
add $form1
```



---

**add layer (to surface)**

Adds the current or specified layer to the current surface, at the end of the layer list.

**Syntax:** add layer|\$lay

**Examples**

```
add layer
add $new_layer
```

---

**add surface (to algorithm)**

Adds the specified surface to the current algorithm.

**Syntax:** add \$srf

**Example**

```
add $add_surface
```

---

**add transform (to layer)**

Adds the specified transform after the current transform in the current layer.

**Syntax:** add \$tra

**Example**

```
add $another_transform
```

---

**add transform point**

Adds a point to the current or specified transform at x, y.

**Syntax:** add transform|\$tra point \$x \$y

**\$x \$y:** Numbers, representing point coordinates

**Example**

```
add transform point 20 40
```

---

**add transform to layer input**

Adds the specified transform after the current transform in the given stream input of the current layer.

**Syntax:** add \$tra to layer input \$count

**\$count:** Number representing layer input number

**Example**

```
$count = 2
add $tra1 to layer input $count
add $tra1 to layer input 2
```

---

## algorithm2 = algorithm1

Sets \$alg2 to refer to \$alg1.

**Syntax:** `$alg2 = $alg`



---

*They refer to the same object, so deleting one of them will cause them both to become invalid. Use the copy command to create separate objects.*

---

## ask action

Defines a button and the action to be carried out if it is selected by a user.

**Syntax:** `Ask Action "Button_Name" [[Close] Goto Label]`

**"Button\_Name":** The text to appear on the button.

**Close:** (Optional). Closes the wizard before carrying out the 'Goto Label'.

**Goto Label:** (Optional). Specifies the label to go to in the script if the user presses the button. If omitted, the button is made inactive and shaded out.

### Example

The following define the buttons used on the first page of a multi-page wizard



---

*In the above example, the "Back" button will be greyed out because no **GoTo Label** is specified.*

---

Similarly, an action button can be put inside the user area of a form, for example:

```
Ask Action "Configure ..." Goto Configure_form
```

There is a 'Close' parameter on the 'Ask Action' command, as well as an explicit "Wizard Close" command. They are designed with different circumstances in mind. The 'Close' parameter on the 'Ask Action' command is primarily designed for quick error pop-up windows, which generally report an error to the user, then pop down, and go straight back to the page that needs more data entered.

---

## ask bandchooser

Creates a chooser dialog which lists the descriptions for all the bands in the specified file.

**Syntax:** `ask bandchooser "prompt text" "title" p_dsh_name [single_band_flag consec_flag] $bandchoice`

**"prompt text"** : The text which appears to the left of the drop down list.

**"title"**: Title of the chooser dialog.

**p\_dsh\_name**: The dataset header file (**.ers**) name.

**single\_band\_flag**: (Optional) Specifies the number of bands that can be selected. Can be either:

- 0 - multiple bands may be selected
- 1 - only one band may be selected

If it is not included it is set to 0 by default. If it is included the `consec_flag` must also be included. If it is set to 1, the `consec_flag` should be set to 0.

**consec\_flag**: (Optional) Automatically selects the both the real and imaginary bands when one or the other is selected. This is designed specifically for specifying bands for FFT. Can be either:

- 0 - has no effect
- 1 - select real and imaginary bands for FFT

If it is not included it is set to 0 by default. If it is included the `single_band_flag` must also be included. If it is set to 1, the `single_band_flag` should be set to 0.



*single\_band\_flag and consec\_flag are optional, but must be used together (i.e. either none or both must be specified)*

**\$band\_choice**: The variable into which the selected band or bands are stored as a string.

### Examples

```
ask bandchooser "Band Menu" "Band Menu Chooser" $filename1 0 1
$BANDSTRING
ask bandchooser "Band Menu" "Band Menu Chooser" $filename1
$BANDSTRING
```



*This stores the choice as a string e.g. 1-3,5,7 and is intended for use with FFT and classification wizards (i.e. wizards that run an executable which takes a band list as an input parameter).*

---

## ask bandmenu

Creates a drop down list of the descriptions for all the bands in the specified file. The band list in the process diagram in the Algorithm dialog is an example of this.

**Syntax:** ask bandmenu "prompt text" p\_dsh\_name \$bandchoice

**"prompt text":** The text which appears to the left of the drop down list.

**p\_dsh\_name:** The dataset header file (.ers) name.

**\$bandchoice:** The band number of the chosen band.

### Example

```
ask bandmenu "My menu" $filename $band_choice
```

---

## ask colorchooser|ask colourchooser

Adds a color chooser button that will open the standard color chooser.

**Syntax:** ask colorchooser "prompt text" "title" \$color

**ask colourchooser "prompt text" "title" \$color**

**"prompt text":** The text which appears to the left of the drop down list.

**"title":** The title of the chooser.

**\$color:** The ColorType variable in which the name of the chosen color is stored.

### Example

```
ask colorchooser "Choose a color" "My color title" $colorchoice
```

---

## ask datum

Adds a file chooser button that will open the standard datum chooser.

**Syntax:** ask datum "label" \$datum\_name

**"label":** The title of the chooser.

**\$datum\_name:** A variable in which the chosen datum from the list is stored.

---

## ask directory

Adds a button that will open a directory chooser.

**Syntax:** ask directory "label" \$dir\_name

**"label":** The title of the chooser.

**\$dir\_name:** A variable in which the chosen directory is stored. You can preset this as a default directory.

### Example

```
$dir_name = "C:\Program Files\ERDAS\ERDAS ER Mapper  
7.2\examples"  
ask directory "Directory:" $dir_name
```

---

---

## ask file

Adds a file chooser button to the alphanumeric entry field.

**Syntax:** `ask file "prompt text" "default_directory" "default_file" ".ext" $file_name`

**"prompt text":** The text appears to the left of the entry box. If you don't want a prompt string use empty quotes: ""

**"default\_directory":** (Optional). The default directory to show in the file chooser, relative to the default directory for the file type. Omit this field if you specify a default file.

**"default\_file":** (Optional). A default file to show in the file chooser. Use the absolute path name or path relative to the default directory for the file type.

**".ext":** A comma separated list of file extensions of the files to be listed in the file chooser. Specify ".ANYRASTER" if you want to list all image files supported by ER Mapper.

**\$file\_name:** A text variable for storing the file name selected using the file chooser.

### Example

```
ask file "Input algorithm or dataset:" ".ers,.alg" $alg_name
```



---

*You can only use the "default\_file" parameter to specify .ers files. To specify another file type as a default, initialize the \$file\_name variable with the default file name. For example:*

```
$outfile = "c:\temp\output.cc8"
ask file "Output dataset: " "" ".cc8" $outfile
```

---

## ask gridlayermenu

Creates a drop down selection list of the descriptions for all the layers in the specified gridding project file.

**Syntax:** `ask gridlayermenu "prompt text" $project_file $layerchoice`

**"prompt text":** The text which appears to the left of the drop down list.

**\$project\_file:** String containing the path and name of the gridding project file.

**\$layerchoice:** The layer number of the chosen layer.

### Example

```
ask gridlayermenu "Grid Layer:" $grid_project_filename
$grid_layer
```

---

---

## ask hardcopy

Asks the user to specify a hardcopy device. The two different types of drivers that are available on the pc platform are `erm_driver` (ermapper hardcopy control files) and `win32_driver` (win32 printer drivers). The `driver_type` option is ignored for unix platforms (`erm_driver` is used regardless of the driver type).

**Syntax:** `ask hardcopy erm_driver|win32_driver "prompt text"`

**erm\_driver:** Add button to open the Default Hardcopy chooser

**win32\_driver:** Add button to open the Windows Print Setup dialog.

**"prompt text":** The text which appears above the entry box.

### Example

```
ask hardcopy win32_driver "Please specify a printer driver"
```

---

## ask link

Adds a dynamic link chooser button to the alphanumeric entry field.

**Syntax:** `ask link "prompt text" chooser_program $var`

**"prompt text":** The text that appears to the left of the entry box. If you don't want a prompt string use empty quotes: ""

**chooser\_program:** The dynamic link chooser program to run. This field is equivalent to the sixth parameter in the dynamic link menu file. See the ["Menu entry parameters"](#) and ["Link chooser parameter"](#) sections.

**\$varL:** A variable for storing the chosen data.

### Example

```
ask link "" "$${CHOOSER=arc_chooser $DEFAULT}" $ws_file
```

---

## ask listmenu

Adds a list containing entries of any defined type.

**Syntax:** `ask listmenu "prompt text" "title" $array_name $choice`

**"prompt text":** The text which appears to the left of the drop down list.

**"title":** The title of the chooser.

**\$array\_name:** The name of the array which holds the choices to be listed in the chooser.

**\$choice:** A variable in which the chosen item from the list is stored.

You can only select one option. You can have lists of the following types (or a mixture of them):

- String
- Value
- LayerType
- Algorithm Mode
- ViewMode
- Coordinate Space Type
- Mosaic Type
- CellType
- TransformType
- SuperSampling Type
- Filter Type
- Color
- Page Constraint Type

The page constraint information is the same as that used in the Page setup dialog.

Most items are listed in the choosers exactly as they are defined in the `$array_name` elements. However, in cases when the resulting lists would not be easy to understand some other property is used instead. The cases this applies to and the property that is listed in the chooser for each case are shown below.

Variable type: Property listed in chooser

Window/Destination: title

Algorithm: name

Stream: desc

Formula: name

Filter: name

Surface: name

You can't have lists of transforms.

If the index for the pointer is null, the item will not show up in the list.

### Example

```
$ARRAY_VARIABLE[1] = red
$ARRAY_VARIABLE[2] = green
$ARRAY_VARIABLE[3] = blue
...
# set the default
$ARRAY_VARIABLE_CHOICE = $ARRAY_VARIABLE[2]
ask listmenu "Choose a layer type from the list" "Layer Chooser"
$ARRAY_VARIABLE $ARRAY_VARIABLE_CHOICE
```

---

### **ask listmenu\_exclusive**

The same as **ask listmenu**, but all the chooser entries are listed in the dialog with exclusive radio buttons (not a drop down list). Thus, this is best used with small numbers of options only.

**Syntax:** `ask listmenu_exclusive "prompt text" $Array_name $choice`

**"prompt text":** The text which appears to the left of the drop down list.

**\$array\_name:** The name of the array which holds the choices to be listed in the chooser.

**\$choice:** A variable in which the chosen item from the list is stored.

---



---

## ask lutmenu

Adds a list of color lookup tables to choose from and stores the selection in a string variable.

**Syntax:** `ask lutmenu "prompt text" $lut_choice`

**"prompt text":** The text which appears to the left of the list.

**\$lut\_choice:** The name of the chosen lookup table is stored in a string variable.

### Example

```
ask lutmenu "Color Lookup Table" $lut_choice
```

---

## ask projection

Adds a file chooser button that will open the standard projection chooser.

**Syntax:** `ask projection "label" $proj_name`

**"label":** The title of the chooser.

**\$proj\_name:** A variable in which the chosen projection from the list is stored.

---

## ask text|number

Adds an alphanumeric entry box to the container.

**Syntax:** `ask text "prompt text" $text_input`

**ask number "prompt text" \$number\_input**

**"prompt text":** The text appears to the left of the entry box. If you don't want a prompt string use empty quotes: ""

**\$text\_input:** A text variable.

**\$number\_input:** A number variable.

### Example:

```
ask text "Please enter your name:"$name
```

---

## ask yes/no

Adds a check box to the container.

**Syntax:** `absolute|relative path`

**"prompt text":** The text appears to the left of the check box. If you don't want a prompt string use empty quotes: ""

**\$yesno\_input:** A Yes/No variable.

### Example:

```
ask yesno "Center Horizontally" $do_center_horiz
```

---

### build absolute filespec

Builds the absolute file specification from a given relative file specification and its parent directory

**Syntax:** `build absolute filespec <parent_dir> <rel_file>`

**parent\_dir:** The path and name of the parent directory

**rel\_file:** The name and path of the file relative to parent\_dir

#### Example

```
println build absolute filespec "c:/Program Files/ERDAS/ERDAS ER
Mapper 7.2/examples" "Data_Types\\Airphoto\\RGB.alg"
```

---

### build file path

Builds a complete path for a file from up to 4 given elements by inserting the correct file separators.

**Syntax:** `build file path <element_1> [<element_2> [<element_3> [<element_4>]]]`

**element\_ :** The path element.

#### Example

```
println build file path "c:\\Program Files\\ERDAS\\ERDAS ER
Mapper 7.2\\examples" "Data_Types" "Airphoto" "RGB.alg"
```

---

### build relative filespec

Builds file specification relative to its parent directory from a given absolute file specification.

**Syntax:** `build relative filespec <parent_dir> <abs_file>`

**parent\_dir:** The path and name of the parent directory

**abs\_file:** The name and absolute path of the file

#### Example

```
println build relative filespec "c:/Program Files/ERDAS/ERDAS ER
Mapper 7.2/examples" "c:\\Program Files\\ERDAS\\ERDAS ER Mapper
7.2\\examples\\Data_Types\\Airphoto\\RGB.alg"
```

---

### Color keywords

You can use the **color\_red**, **color\_green** and **color\_blue** keywords to get the rgb (0-255) components of a color variable.

#### Example

```
$bg_color = 230,23,56
$red = get $bg_color color_red
$blue = get $bg_color colour_blue
```

---

**container right|left justify**

This is used to justify a button or a row of buttons in a container. It will not affect other item types such as lists and files as these are placed so as to make best use of the existing space.

**Syntax:** [Container] right|left justify

**Example**

```
container right justify
  ask action "< Back"
  ask action "Next >"
```

---

**container above | below | left | right**

Optional command specifies how this container is to be placed in relation to another container. The default is for the container to be below the previously defined container.

**Syntax:** [Container] Above | Below | Left | Right "Name"

**"Name":** String with name of other container

**Example**

```
container below "con1"
```

---

**container begin|end**

A container block defines the size, contents and layout of a single 'container' or 'pane' in a wizard page. Any number of containers can be defined though realistically there will be only two or three per page.

**Syntax:** container begin "Name"

.....  
**container end**

**"Name":** Name is the name of the container within the script. Each of the containers on a single page (within a single Wizard block) must have a different name.

**Example**

```
container begin "con2"
  container height_pct 20
  container below "con1"
    ask action "< Back"
    ask action "Next >" goto wizard_page2
    ask action "Cancel" close goto wizard_page2
  container end
```

---

## container items

Optional command specifies the direction of placement of the *items in a container* defined between this command and the next Container Items command. If omitted, the items will be placed vertically.

**Syntax:** [Container] [Items] Horizontal | Vertical | Newline

### Examples

```
Container Items Horizontal
Horizontal
```

---

## container labels

Optional command specifies where the labels for an item in the container will appear. The default is 'above'.

**Syntax:** [Container] Labels\_above | Labels\_left

### Example:

```
Container labels_above
```

---

## container width|height

Optional command specifies the container width and height as a percentage of the remaining space.

**Syntax:** [Container] width\_pct | height\_pct \$Percentage\_number

**\$Percentage\_number:** Number, representing percentage value.

### Examples

```
Container width_pct 30
width_pct 30
```

---

## convert file separator

Converts the file separators in a given file specification from English to native and vice versa.

**Syntax:** convert <filespec> to english|native sep|separator

**filespec:** File specification to be converted.

### Example

```
$fspec = "c:/Program Files/ERDAS/ERDAS ER Mapper
7.2/examples/Data_Types/Airphoto/RGB.alg"
convert $fspec to native sep
```

---

**copy algorithm**

Copies the current or specified algorithm. The current algorithm pointer is updated to point to the new algorithm.

**Syntax:** `copy algorithm|$alg`

**Examples**

```
copy algorithm to window
copy $alg to window
$alg_copy = copy $alg # Note: not the same as $alg_copy = $alg
$alg_copy = copy algorithm
```

---

**copy algorithm from window**

Copies the algorithm from the current or specified window to the batch engine and set the current algorithm pointer to point to it.

**Syntax:** `copy algorithm from window|$win`

**Examples**

```
copy algorithm from window
copy algorithm from $win
```

---

**copy algorithm to window**

Copies the specified algorithm or that indicated by the current algorithm pointer to the current window in ER Mapper. Aborts if there is no current window

**Syntax:** `copy algorithm|$alg to window`

**Examples**

```
copy $original_algorithm to window
copy algorithm from window
```

---

**copy filter**

Makes a copy of the current or specified filter but does not insert the new filter into a layer.

**Syntax:** `copy filter|$fil`

**Example**

```
$fil = copy filter
```

---

**copy formula**

Makes a copy of the current or specified formula, but does not add it to a layer.

**Syntax:** `copy formula|$for`

**Example**

```
$for2 = copy formula
```

---

**copy layer**

Copies the current or specified layer but does not insert it into a surface.

**Syntax:** copy layer|\$lay

**Examples**

```
$temp_layer = copy layer
```

---

**copy surface**

Copies the current or specified surface, but does not add the new surface to any algorithm

**Syntax:** copy surface|\$srf

**Example**

```
$new_surface = copy $srf1
```

---

**copy transform**

Makes a duplicate of the current or specified transform but does not insert it into the current algorithm.

**Syntax:** copy transform|\$tra

**Examples**

```
copy $tra1  
$tra2 = copy $tra1
```

---

**copy window**

Makes a duplicate of the current or specified window and its algorithm, and then runs the algorithm to display the duplicate. The current window pointer is updated to point to the new window.

**Syntax:** copy window|\$win

**Examples**

```
copy window  
copy $win
```

---

**current algorithm**

Sets the pointer to the current algorithm. Can also be used to check whether an algorithm currently exists

**Syntax:** current algorithm

**Examples**

```
$curr_alg = current algorithm  
if current algorithm then goto have_algorithm
```

---

**current filter**

Sets the pointer to the current filter.

**Syntax: current filter**

**Example**

```
$fil = current filter
```

---

**current formula**

Sets the pointer to the current formula.

**Syntax: current formula**

**Example**

```
$for1 = current formula
```

---

**current layer**

Sets the pointer to the current layer. This can also be used to check whether a current layer exists.

**Syntax: current layer**

**Examples**

```
$layer2 = current layer  
if current layer then goto layer_ok
```

---

**current surface**

Sets the pointer to the current surface.

**Syntax: current surface**

**Example**

```
$curr_surface = current surface
```

---

**current transform**

Sets the pointer to the current transform.

**Syntax: current transform**

**Example**

```
$tral = current transform
```

---

**current window**

Sets pointer to the current window. This can also be used to check whether an image window currently exists.

**Syntax: current window**

**Examples**

```
$win = current window
if current window then goto window_ok
```

---

**delete algorithm**

Deletes all references to an algorithm. If the algorithm deleted was the current algorithm, the current algorithm pointer will be set to point to the next algorithm if one exists, or the previous one, or aborts.

**Syntax: delete algorithm|\$alg**

**Examples**

```
delete algorithm
delete $change_algorithm
```

---

**delete directory|file**

Delete a directory or file. If the file name has an **“.ers”** extension, the raster file is also deleted.

**Syntax: delete \$dir|\$file**

**\$dir**: String with path and directory name

**\$file**: String with path and file name

**Example**

```
$temp_file = "examples/tempfile.ers"
delete $temp_file
```

---

**delete filter**

Deletes the filter and all references to it.

**Syntax: delete filter|\$fil**

**Example**

```
delete filter
```

---

**delete formula**

Deletes the current formula in the current layer.

**Syntax: delete formula**

**Example**

```
delete formula
```

---



---

**delete layer**

Deletes the current or specified layer. When all layers in a surface are deleted, the surface will also be deleted unless it is the only surface on the algorithm.

**Syntax:** delete layer|\$lay

**Examples**

```
delete $layer1
delete layer
```

---

**delete status (dialog)**

Deletes the status dialog. Note that there is only one status dialog for the batch engine.

**Syntax:** delete status

**Example**

```
delete status
```

---

**delete surface**

Deletes the current or specified surface from the current algorithm.

**Syntax:** delete surface|\$srf

**Examples**

```
delete $add_surface
delete surface
```

---

**delete transform**

Deletes the current or specified transform, and all references to it.

**Syntax:** delete transform|\$tra

**Example**

```
delete $tra1
```

---

**delete window**

Deletes the current or specified window and its algorithm. If the one deleted was current it updates the current window and current algorithm to the next window and next algorithm. If there is no next window it sets them to the previous window and previous algorithm.

**Syntax:** delete window|\$win

**Examples**

```
delete window
delete $win
```

---

**duplicate filter**

Copies the current filter, and inserts the copy after the current filter.

**Syntax:** `duplicate filter`

**Examples**

```
duplicate $fil
$fill = duplicate filter
```

---

**duplicate formula**

Duplicates the current formula, and inserts the copy after the current formula.

**Syntax:** `duplicate formula`

**Examples**

```
$form1 = duplicate formula
duplicate layer
```

---

**duplicate layer**

Duplicates the current layer in the current surface. The new layer is inserted after the current layer.

**Syntax:** `duplicate layer`

**Examples**

```
$lay1 = duplicate layer
duplicate layer
```

---

**duplicate surface**

Duplicates the current or specified surface within the current algorithm.

**Syntax:** `duplicate surface|$srf`

**Examples**

```
$new_surface = duplicate surface
duplicate surface
```

---

**duplicate transform**

Copies the current transform in the current layer, and inserts the copy after the current transform.

**Syntax:** `duplicate transform`

**Example**

```
duplicate transform
```

---

## edit file

Edit the given file name in a text editor. The file is created if it doesn't exist.

**Syntax: edit \$filename**

**\$filename:** String with path and text file name, including extension.

**Example**

```
$toolbarname = "c:\Program Files\ERDAS\ERDAS ER Mapper
7.2\batch\toolbar.erb"
edit $toolbarname
```

---

## exists

Verify that a specified file exists.

**Syntax: \$filename exists**

**\$filename:** String with path and name of file.

**Example**

```
if "c:\temp\temp.alg" exists then goto file_exists
```

---

## exit

Exit the batch process and return to ER Mapper.

**Syntax: exit [\$exit\_no]**

**\$exit\_no:** Integer with return code. Only used if you are running the batch script from the command line using **ermapper -b**. The script must exit with a return code > 0 (e.g. `exit 1`) to ensure that the ER Mapper application also exits. If the script exits with `exit` or `exit 0`, then the command line prompt will "hang" until you physically stop the ER Mapper application.

**Example**

```
if ($var == 1) then goto carryon
exit
carryon:
```

---

## File name, extension and path keywords

You can use the **filename**, **dirname** and **fileext** keywords to respectively return the file name, path and extension of a given file specification. If there is no extension, filename or path component it returns a null string ""

### Examples

```
$string = "\examples\Shared_Data\airphoto.ers"  
$fname = filename $string# returns "airphoto.ers"  
$dirname = dirname $string# returns "examples\Shared_Data"  
$ext = fileext $string# returns ".ers"
```



*If \$string = "c:\", then dirname \$string would return "c:\"; i.e it retains the trailing slash for volumes.*



*If \$string = "/examples/Shared\_Data/airphoto.ers" then dirname \$string would return "examples/Shared\_Data".*

---

## filter2 = filter1

Assigns specified filter (e.g. \$fil1) to new variable (e.g. \$fil2)

**Syntax: \$fil2 = \$fil1**



*\$fil2 and \$fil1 refer to the same object, so deleting \$fil1 will invalidate \$fil2. Use the copy or duplicate command to create a new object.*

### Example

```
$fil2 = $fil1
```

---

## first|last|next|previous algorithm

Changes the current algorithm pointer to point to the first, last, next or previous algorithm.

**Syntax: first|last|next|previous algorithm**

### Examples

```
next algorithm  
$alg2 = last algorithm
```

---

**first|last|next|previous filter**

Sets the current filter to the first, last, next or previous filter in the current layer. Optionally selects an input filter, and/or a filter type.

**Syntax:** first|last [input \$n] [\$ftype] filter

next|previous [\$ftype] filter

**\$n:** Number, representing input number

**\$ftype:** Filter type variable: value can be convolution, threshold or user.

**Example**

```
last input 2 filter
next threshold filter
```

---

**first|last|next|previous formula**

Sets the current formula to the first, last, next or previous formula in the current layer.

**Syntax:** first|last formula

next|previous formula

**Example**

```
last formula
next formula
```

---

**first|last|next|previous layer**

Sets the current layer pointer to point to the specified layer in the current surface. Sets \$ERROR to 1 if fails; otherwise 0.

**Syntax:** first|last|next|previous layer

**Examples**

```
$layer1 = first layer
first layer
```

---

**first|last|next|previous transform**

Sets the current transform to the first, last, next or previous transform in the current layer. Optionally select an input transform, and/or a transform type.

**Syntax:** **first|last [input \$n] [\$ttype] transform**  
**next|previous [\$ttype] transform**

**\$n:** Number representing layer input number

**\$ttype:** Transform type variable: value can be linear, exp, log or hist

**Examples**

```
last transform
$lasttran = last input 2 transform
next linear transform
$nextlin = next linear transform
```

---

**first|last|next|previous window**

Updates the current window pointer to point to the oldest, newest, next oldest or next newest window open. Sets \$ERROR to 1 if fails; otherwise 0.

**Syntax:** **first|last|next|previous window**

**Examples**

```
next window
$winlast = last window
```

---

**first|last|next|previous surface**

Selects a surface within the current algorithm and makes it current.

**Syntax:** **first|last|previous|next surface**

**Examples**

```
first surface
next surface
```

---

**fit page to hardcopy**

Sets the page width and height of the current or specified algorithm to that of the currently specified hardcopy device.

**Syntax:** **fit [\$alg] page to hardcopy**

**Example**

```
fit $alg page to hardcopy
```

---

**format value precision**

Formats a number to have a specified number of digits after the decimal point.

**Syntax:** `format $input_value $precision`

**\$input\_value:** Number, representing original data to be formatted

**\$precision:** Number, representing required number of digits after the decimal point.

**Example**

```
$arg1_formatted = format $template_tl_e_extent 3
```

---

**get (page) contents topleft|bottomright**

Gets the page contents extents coordinates.

**Syntax:** `get [$alg] contents topleft|bottomright northings|latitude|meters_y`

`get [$alg] contents topleft|bottomright eastings|longitude| meters_x`

**Example**

```
$template_tl_e_extent = get $alg contents topleft eastings
```

---

**get algorithm coordsys**

Gets the algorithm's coordinate system type.

**Syntax:** `get algorithm|$alg coordsys`

**Examples**

```
$coordsys = get algorithm coordsys
```

```
$coordsys = get $alg coordsys
```

---

**get algorithm datum|projection**

Gets the algorithm's datum or projection type.

**Syntax:** `get algorithm|$alg datum|projection`

**Examples**

```
$datum = get algorithm datum
```

```
$proj = get $alg projection
```

---

**get algorithm description**

Gets the current or specified algorithm's description.

**Syntax:** `get algorithm|$alg description`

**Examples**

```
$desc = get algorithm description
```

```
$desc = get $alg description
```

---

---

**get algorithm layer count**

Gets the number of layers in the current or specified algorithm.

**Syntax:** `get algorithm|$alg layer count`

**Examples**

```
$count = get algorithm layer count
$count = get $new_alg layer count
```

---

**get algorithm lut (color table)**

Gets the algorithm's pseudocolor LUT name.

**Syntax:** `get algorithm|$alg lut`

**Example**

```
$alg_lut = get algorithm lut
```

---

**get algorithm mode**

Gets the algorithm's mode. Returns "pseudo", "rgb" or "hsi".

**Syntax:** `get algorithm|$alg mode`

**Examples**

```
$algorithm_mode = get algorithm mode
if (get algorithm mode == pseudo) then goto newmode
```

---

**get algorithm mosaic type**

Gets the algorithm's mosaic type. Returns overlay or feather.

**Syntax:** `get algorithm|$alg mosaic type`

**Example**

```
$mtype = get algorithm mosaic type
```

---

**get algorithm topleft|bottomright (extents)**

Gets the algorithm's extents. Returns number.

**Syntax:** `get algorithm|$alg topleft|bottomright eastings| longitude|meters_x`  
`get algorithm|$alg topleft|bottomright northings| latitude|meters_y`

**Example**

```
$extent1 = get algorithm bottomright longitude
```



---

**get algorithm units|rotation**

Gets the algorithm's units or rotation. Rotation is in decimal degrees, units is a units string.

**Syntax:** `get algorithm|$alg units|rotation`

**Examples**

```
$alg_units = get algorithm units
$alg_rotate = get $alg rotation
```

---

**get English file separator**

Returns the standard English file separator used by the host workstation or PC. This would be "\" on a Win32 (PC) platform or "/" on a Unix platform.

**Syntax:** `get english file sep|separator`

**Example**

```
println get english file sep
```

---

**get file size**

Gets the given files size in bytes. \$filename is an absolute path name.

**Syntax:** `get $filename size`

**\$filename:** String with path and file name

**Example**

```
$filename = "c:\Program Files\ERDAS\ERDAS ER Mapper
7.2\examples\myfile.alg"
$filesize = get $filename size
```

---

**get filter description**

Gets the current or specified filter's description.

**Syntax:** `get filter|$fil description`

**Examples**

```
$desc = get filter description
$desc = get $fil1 description
```

---

**get (usercode) filter params**

Gets the current or specified user filter's parameters.

**Syntax:** `get filter|$fil params`

**Example**

```
$param = get $filter1 params
```

---

**get filter postsampled (flag)**

Gets the current or specified filter's post sampled process flag. Returns boolean value.

**Syntax:** `get filter|$fil postsampled`

**Example**

```
$fpost = get filter postsampled
```

---

**get filter rows|cols**

Gets the number of rows/columns in the current or specified filter.

**Syntax:** `get filter|$fil rows|cols`

**Examples**

```
$frow = get filter rows  
$fcol = get $fil1 cols
```

---

**get filter scale|threshold (value)**

Gets the scale or threshold for the current or specified filter. Valid only on convolution and threshold filters.

**Syntax:** `get filter|$fil scale|threshold`

**Example**

```
$value = get filter threshold
```

---

**get filter userfile (filename)**

Gets the current or specified user filter source file name. Valid only on usercode filters.

**Syntax:** `get filter|$fil userfile`

**Example**

```
$fnme = get filter userfile
```

---

**get (usercode) filter userfunc (filename)**

Gets the current or specified user filter function (.dll) name. Valid only on usercode filters.

**Syntax:** `get filter|$fil userfunc`

**Example**

```
$funcname = get filter userfunc
```

---

**get filter type**

Gets the current or specified filter's type. Returns convolution, threshold or user.

**Syntax:** `get filter|$fil type`

**Example**

```
$ftype = get filter type
```

---

**get free space (on disk)**

Gets the amount of free space, in bytes, on the given disk.

**Syntax:** `get $diskname free space`

**\$diskname:** String with path and directory name

**Example**

```
$diskname = "c:\Program Files\ERDAS\ERDAS ER Mapper
7.2\examples"
$free_space = get $diskname free space
```

---

**get layer azimuth|elevation**

Gets the layer sunshading azimuth or elevation. Returns degrees as a number.

**Syntax:** `get layer|$layazimuth|elevation`

**Example**

```
$angle = get layer elevation.
```

---

**get layer band count**

Gets the number of bands in the current or specified layer.

**Syntax:** `get layer|$lay band count`

**Example**

```
$band_count = get layer band count
```

---

**get layer band description**

Gets the nominated band description of the current or specified layer.

**Syntax:** `get layer|$lay band $n description`

**\$n:** Number, representing band number

**Example**

```
$band_desc = get layer band 1 description
```

---

**get layer cell sizex|sizey**

Reads the current or specified layer cell horizontal or vertical size.

**Syntax:** `get layer|$lay cell sizex|sizey`

**Examples**

```
$cell_sizex = get $layer1 cell sizex
$cell_sizey = get layer cell sizey
```

---

**get layer cell type**

Gets the current or specified layer's image cell type.

**Syntax:** `get layer|$lay cell type`

**Example**

```
$lay_celltype = get layer cell type
```

---

**get layer color**

Gets the current or specified layer color. Valid only on link layers.

**Syntax:** `get [layer|$lay] color`

**Examples**

```
$layer_color = get color  
$layer_color = get $layer color
```

---

**get layer coordinates**

Returns the EN or LL layer coordinates from the given cellX and cellY values.

**Syntax:** `get layer x_coordinate|y_coordinate from $cellX $cellY`

\$cellX \$cellY: Integers with X and Y cell coordinate values

**Examples**

```
$tlx = get layer x_coordinate from $StartColumn $StartRow  
$tly = get layer y_coordinate from $StartColumn $StartRow  
$brx = get layer x_coordinate from $EndColumn $EndRow  
$bry = get layer y_coordinate from $EndColumn $EndRow
```

---

**get layer dataset (filename)**

Gets the current or specified layer's image name.

**Syntax:** `get layer|$lay dataset`

**Examples**

```
$dsname = get layer dataset  
if (get layer dataset != "") then goto make_cdrape
```

---

**get layer description**

Gets the current or specified layer's description.

**Syntax:** `get layer|$lay description`

**Example**

```
$lay_desc = get $layer1 description
```

---

**get layer editable (flag)**

Gets the current or specified layer's editable flag. Valid only on link layers. Returns true or false.

**Syntax:** `get layer|$lay editable`

**Example**

```
$edit = get layer editable
```

---

**get layer edit|init program (name)**

Gets the current or specified layer's edit|init program name.

**Syntax:** `get layer|$lay edit|init program`

**Example**

```
$init_program = get layer init program
```

---

**get layer formula**

Gets a copy of the current or specified layer's formula. Returns a structure useable by formula commands but not println.

**Syntax:** `get layer|$lay formula`

**Example**

```
$form = get $layer1 formula
```

---

**get layer input filter count**

Gets the number of input filters in the current or specified layer input.

**Syntax:** `get layer|$lay input $index filter count`

**\$index:** Number representing input number.

**Example**

```
$count = get layer input $index filter count
```

---

**get layer link extension**

Gets the link file extension string for the current or specified layer.

**Syntax:** `get layer|$lay link extension`

**Examples**

```
$l_ext = get $layer1 link extension  
if (get layer link extension == ".erv") then goto process_vector
```

---

**get layer output transform count**

Gets the number of output transforms in the current or specified layer.

**Syntax:** `get layer|$lay output transform count`

**Example**

```
$count = get $layer1 output transform count
```

---

**get layer shading**

Gets the current or specified layer's shading value. Returns on or off

**Syntax:** `get layer shading`

**Example**

```
$shaded = get layer shading
```

---

**get layer type**

Gets the current or specified layer's type. Returns a layer type value. See ["Variables"](#).

**Syntax:** `get layer|$lay type`

**Example**

```
$ltype = get layer type
```

---

**get layer|\$lay input count**

Gets the number of inputs in the current or specified layer.

**Syntax:** `get layer|$lay input count`

**Example**

```
$count = get layer input count
```

---

**get layer visibility**

Determines if any part of the current or specified layer is turned on and lies within the extents of the algorithm.

**Syntax:** `get layer|$lay visibility`

**Example**

```
$layer_visible = get $layer1 visibility  
$layer_visible = get layer visibility
```

---

**get native path|file separator**

Get the path or file separator used natively by the host workstation or PC.

**Syntax:** `get native path|file sep|separator`

**Example**

```
println get native path separator
println get native file sep
```

---

**get page constraints**

Get the page constraints of the current or specified algorithm. Returns zoom, page, border or scale.

**Syntax:** `get [$alg] page constraints`

**Example**

```
$page_constraint = get $alg page constraints
```

---

**get page scale**

Gets the page scale of the current or specified algorithm.

**Syntax:** `get [$alg] page scale`

**Example**

```
$page_scale = get $alg page scale
```

---

**get page size**

Gets the page size of the current or specified algorithm. Returns string with page size; e.g. "US Letter".

**Syntax:** `get [$alg] page size`

**Example**

```
$page_size = get $alg page size
```

---

**get page topleft|bottomright (extents)**

Gets the page extents coordinates.

**Syntax:** `get [$alg] page topleft|bottomright northings|latitude|meters_y`

`get [$alg] page topleft|bottomright eastings|longitude| meters_x`

**Example**

```
$tl_e_extent = get $alg page topleft eastings
```

---

**get page top|bottom|left|right border**

Gets the page borders of the current or specified algorithm.

**Syntax:** `get [$alg] page top|bottom|left|right border mm|inches`

**Example**

```
$page_top = get $alg page top border inches
```

---

**get (algorithm) page view mode**

Gets the page view mode (normal or page layout) of the current or specified algorithm.

**Syntax:** `get [$alg] page view mode`

**Example**

```
$pvmode = get page view mode
```

---

**get page width|height**

Gets the specified page parameter of the current or specified algorithm,

**Syntax:** `get [$alg] page width|height inches|mm`

**Example**

```
$page_width = get $alg page width mm
```

---

**get preference**

Gets the value of the preference name from the preference file. Returns specified default value if entry does not exist.

**Syntax:** `get preference "name" $default_string|$default_number[boolean|integer|double]`

**"name":** String with name of entry in preference file. Suggested standard format is "Class:Name:Variable".

**\$default\_string:** String with default value for entry

**\$default\_number:** Number with default value for entry

**Example**

```
$ImageVersion = get preference "Wizard:Image:Version" 1.0
```



---

## get preference (color)

Gets the color preference entry in the user's preference file. Set to specified default value if the entry does not exist.

**Syntax:** `get preference "colorname" $red $green $blue`

`get preference "colorname" colorval $default_color|$red,$green,$blue`

**"colorname"**: Color preference entry name

**\$red \$green \$blue**: RGB numeric values for the default color.

**\$default\_color**: Default color specification in the form of a string (e.g. "red"), an RGB triple (e.g. 255,0,0 for red), or a variable which has been set to a color specification.

### Examples

```
$pref_color = get preference "my_color_pref" $default_color
$pref_color = get preference "my_color_pref" colorval
$default_color
$pref_color = get preference "my_color_pref" colorval "green"
$pref_color = get preference "my_color_pref" colorval
100,203,240
$pref_color = get preference "my_color_pref" 100 203 240
$pref_color = get preference "my_color_pref" 100,203,240
```

---

## get surface description

Gets the current or specified surface description.

**Syntax:** `get surface|$srf description`

### Examples

```
$desc = get surface description
$desc = get $srf1 description
```

---

## get surface layer count

Gets the number of layers in the current or specified surface.

**Syntax:** `get surface|$srf layer count`

### Examples

```
$layers = get surface layer count
$layers = get $srf layer count
```

---

## get transform (limits)

Gets the current or specified transform's limits. Returns a number.

**Syntax:** `get transform|$tra input|output min|max`

### Example

```
$inmax = get transform input max
```

---

### get transform type

Gets the current or specified transform type. Returns linear, exp, log or hist.

**Syntax:** `get transform|$tra type`

#### Example

```
$strantype = get transform type
```

---

### get (algorithm) view mode

Gets the view mode of the current or specified algorithm.

**Syntax:** `get [$alg] view mode`

#### Example

```
$vmode = get view mode
```

---

### get (ER Mapper) version

Returns the ER Mapper version number as a string or as a number.

**Syntax:** `get version_string|version_number`

#### Examples

```
$version_string = get version_string# returns "5.7"
```

---

### getenv (environment variable)

Get the given environment variable. This is from the current environment within ER Mapper.

**Syntax:** `getenv $envname`

**\$envname:** String with name of environment variable

#### Examples

```
$machine_type = getenv "ERM_MACHINE_TYPE"  
$println "ERMSCRIPTS =" + getenv "ERMSCRIPTS"
```

---

### go algorithm

Runs the current or specified algorithm. This causes histograms etc. to be updated.

**Syntax:** `go algorithm|$alg [width height][match]`

**width height:** Resolution at which to run algorithm (default 400 400)

**match:** If specified, runs a histogram match.

#### Examples

```
go algorithm  
go algorithm 50 50  
go $alg1 100 100  
go algorithm 400 400 match
```

---

## go background window

Runs the algorithm in the current or specified window as a background task.; i.e performs a non-blocking go on the window.

**Syntax:** go background window|\$win

### Examples

```
go background window
go background $win
```

---

## go window

Runs the algorithm in the current or specified window.

**Syntax:** go window|\$win

### Examples

```
go window
go $win
```

---

## layer active

Checks whether the current layer is active, and returns TRUE (1) for active and FALSE (0) for inactive

**Syntax:** layer active

### Examples

```
$layer_is_on = layer active
```

---

## layer2 = layer1

Assigns specified layer (e.g. \$lay1) to another variable (e.g. \$lay2)



*\$lay2 and \$lay1 refer to the same object, so deleting \$lay1 will invalidate \$lay2. Use copy or duplicate layer to have different objects.*

**Syntax:** \$lay2 = \$lay1

### Example

```
$lay2 = $lay1
```

---

## listdir

Lists the contents of the specified directory into an array, with each element being one file. If a recursive listing is made, then the resultant strings will contain the full path, otherwise they will just be the filename.

**Syntax:** `listdir $directory [$extension] [recursive]`

**\$directory:** String defining the directory to list

**\$extension:** Optional string limiting the list to files with a specific extension (eg. `*.ers`)

**recursive:** Specify this keyword to descend recursively into sub-directories, and list all files in the directory tree. This returns filenames with their full path.

### Examples

```
$array = listdir "e:\images" "*.ers" recursive
```

---

## load algorithm

Loads the given algorithm. Sets the current algorithm pointer to point to it. Sets \$ERROR to 1 if it fails; otherwise 0.

**Syntax:** `load algorithm $name`

**\$name:** Algorithm filename; e.g. `myalg.alg`. Must be specified relative to the current directory or include the absolute path.

### Examples

```
$template_alg_name = "c:\Program Files\ERDAS\ERDAS ER Mapper  
7.2\examples\templates\some.alg"  
$template_alg_name = load algorithm $template_alg_name  
$myfile = "..\..\examples\tutorial\my.alg"  
load algorithm $myfile
```

---

## load filter (filename)

Loads the current or specified filter from the given (.ker) file. The file name must include the absolute or relative path.

**Syntax:** `load filter|$fil $filename`

**\$filename:** String with path and name of filter (.ker) file

### Examples

```
$filename = "filters\myfilter.ker"  
load filter $filename  
load $fil "filters\myfilter.ker"
```

---

### load formula (filename)

Loads the current or specified formula file.

**Syntax:** load formula|\$for from \$fname

**\$fname:** String, with path and name of formula (.frm) file

#### Examples

```
load formula from "ratio\clay_ratio.frm"
$for = load formula from "ratio\clay_ratio.frm"
```

---

### load layer formula (filename)

Loads the formula file into the current or specified layer's formula.

**Syntax:** load layer|\$lay formula \$fname

**\$fname:** String, with formula filename (.frm) and path.

#### Example

```
$form = "ratio\clay_ratio.frm"
load $layer1 formula $form
```

---

### match transform [to \$layer]

Matches the output transforms of all layers to the current or specified layer in the same surface.

The algorithm must already contain output transforms for the layers being matched. Use the slower **go algorithm match** command if the transforms do not exist.

**Syntax:** match transform [to \$layer]

#### Examples

```
$temp_layer = first active raster layer
match transform to $temp_layer
first active raster layer
match transform
```

---

### move layer

Moves the current or specified layer within the current surface to the given position within its surface.

**Syntax:** move layer|\$lay [to] up|down|top|bottom

#### Examples

```
move layer to top
move $layer up
```

---

**move surface**

Moves the current surface within the current algorithm.

**Syntax:** `move surface up|down|top|bottom`

**Example**

```
move surface bottom
```

---

**new algorithm**

Creates a new (empty) algorithm. This will have 1 pseudocolor layer. The current algorithm pointer is updated to point to the new algorithm

**Syntax:** `new algorithm`

**Examples**

```
new algorithm
$alg = new algorithm
```

---

**new filter**

Creates a new filter, but does not add it to any layer.

**Syntax:** `new $ftype|convolution|threshold|user filter`

**\$ftype** Filter type variable: value can be convolution, threshold or user.

**Example**

```
$fill = new convolution filter
```

---

**new formula**

Creates a new formula.

**Syntax:** `new formula`

**Example**

```
$for1 = new formula
```

---

## new layer

Creates a new layer of the given type. This creates a floating layer which does not belong to any surface in any algorithm. To add it to a surface use `add layer`. This could be useful if you want to create a layer and duplicate it and change the processing before adding it.

**Syntax:** `new $layer_type|pseudocolor|red|green|blue|hue|saturation|intensity|height|classification|classdisplay|link layer`

**\$layer\_type:** Layer type variable: see ["Variables"](#) for allowed values.

### Examples

```
$ltype = height
new $ltype layer
new pseudocolor layer
$new_layer = new pseudocolor layer
```

---

## new surface

Creates a new surface but does not add it to any algorithm.

**Syntax:** `new surface`

### Examples

```
new surface
$add_surface = new surface
```

---

## new transform

Creates a new transform which then becomes the current transform.

**Syntax:** `new $ttype|linear|exp|log|hist transform`

**\$ttype:** Transform type variable: value can be linear, exp, log or hist

### Examples

```
new linear transform
%tran2 = new linear transform
```

---

## new window

Opens a new image window, with a default algorithm. The current window pointer points to the new window. The current algorithm pointer points to the new algorithm. You can specify the position and size of the window.

**Syntax:** `new window [x y w h]`

**x, y:** The x,y coordinate of the top left corner of the window in screen pixels.

**w, h:** The width and height of the window in screen pixels.

### Examples

```
new window
$win = new window 0 0 600 600
new window $Xoffset $Yoffset $windowwidth $windowheight
```

---

### next [active][raster|vector] layer (type)

Moves the pointer to the next layer of the type specified. With layers there are the additional keywords 'next' and 'active'.

**Syntax:** next [active][raster|vector] [\$layer\_type] layer

**\$layer\_type:** Layer type variable: see ["Variables"](#) for allowed values.

#### Examples

```
next layer
next active layer
next active raster layer
next active vector layer
next red layer
```

---

### open status (dialog)

Creates (if necessary) and opens the status dialog. This is useful for when the user has pressed the close button on the status dialog. If "title" is included it is used as the title of the progress dialog.

The wizard GUI may have a button labelled **Status**.

**Syntax:** open status ["title"]

**"title":** The text which appears in the title bar of the status dialog box.

#### Example

```
...
ask action "Status" goto OpenStatus
...
OpenStatus:
open status
goto wizard_page_1
```

---

### open window

Open the designated dialog box on screen. If the dialog box is iconised, it will be un-iconised.

**Syntax:** open main|algorithm|transform|filter|formula| sunangle|page  
setup|annotation|defaults|preferences|  
geoposition|processinfo|datasetinfo|scattergram|  
traverse|realtime3d|job|profile|cell coordinate window

#### Example

```
open sunangle window
```



---

## print

Write to an output dialog (default) or file without a Carriage Return or Line Feed.

**Syntax:** `print "String"|Number|$var`

**\$var:** Variable with string or number. By default, numbers are printed to 6 decimal places.

The `print` and `println` commands write to an output dialog by default. Alternatively, you can print out to a file using:

```
set output to $filename
```

The `print` or `println` commands in a batch script after this `set` command will create the file if it doesn't already exist and write the output to the end of the file.

### Example

```
$text="Hello World"  
print $text  
print "Hello World"
```

---

## println

Write to an output dialog (default) or file with a Carriage Return or Line Feed.

**Syntax:** `println "String"|Number|$var`

**\$var:** Variable with string or number. By default, numbers are printed to 6 decimal places.

The `print` and `println` commands write to an output dialog by default. Alternatively, you can print out to a file using:

```
set output to $filename
```

The `print` or `println` commands in a batch script after this `set` command will create the file if it doesn't already exist and write the output to the end of the file.

### Example

```
$text="Hello World"  
println $text  
println "Hello World"
```

---

## save algorithm

Saves the current or specified algorithm with the given name. Sets \$ERROR to 1 if fails; otherwise 0.

### Syntax: save algorithm|\$alg \$name

**\$name:** Algorithm filename; e.g. myalg.alg. Must be specified relative to the current directory or include the absolute path.

### Examples

```
$alg_name = "c:\Program Files\ERDAS\ERDAS ER Mapper
7.2\examples\templates\some.alg"
save algorithm $alg_name
save $alg $alg_name
```

---

## save algorithm as dataset

Saves the current or specified algorithm as a dataset, with the name stored in the variable \$dsname. If the optional parameters are not set, the default values which would display in the Save as Dataset dialog will be used.

### Syntax: save algorithm|\$alg as dataset \$dsname [\$celltype \$null\_value \$nr\_cells \$nr\_lines]

**\$dsname:** String representing the path and name of dataset (.ers) file.

**\$cell\_type:** Cell type variable: value can be uint8, uint16, uint32, int8, int16, int32, ieee4 or ieee8. e.g \$cell\_type = uint16 (not "uint16")

**\$null\_value:** String representing the null\_value to be used. It may be set to "none".

**\$nr\_cells:** Number representing the number of columns of the image to include.

**\$nr\_lines:** Number representing the number of rows of the image to include.

### Examples

```
save $alg as dataset $dsname
$celltype = u16int
save $alg as dataset $dsname $celltype $null_value $nr_cells
$nr_lines
save as dataset $dsname
save algorithm as dataset $dsname $celltype $null_value
$nr_cells $nr_lines
save $alg as dataset $dsname $celltype "none" $nr_cells
$nr_lines
```

---

### save algorithm as virtual dataset

Saves the current or specified algorithm as a virtual dataset with the given name. Sets \$ERROR to 1 if it fails; otherwise 0.

**Syntax:** save algorithm|\$alg as virtual dataset \$name

**\$name:** Virtual dataset filename; e.g. **myvds.ers**. Must be specified relative to the current directory or include the absolute path.

#### Examples

```
$vds_name = "c:\Program Files\ERDAS\ERDAS ER Mapper
7.2\examples\tutorial\somevds.ers"
save algorithm as virtual dataset $vds_name
save $alg as virtual dataset $vds_name
```

---

### save filter (filename)

Saves the current or specified filter to the given (.ker) file.

**Syntax:** save filter|\$fil \$fname

**\$fname:** String with path and name of filter (.ker) file

#### Example

```
$fname = "filters\myfilter.ker"
save filter $fname
```

---

### save formula (filename)

Saves the current or specified formula to the formula file.

**Syntax:** save formula|\$for to \$fname

**\$fname:** String, with path and name of formula (.frm) file

#### Example

```
save formula to "ratio\clay_ratio.frm"
```

---

### save layer formula (filename)

Saves the current or specified layer's formula to the formula file.

**Syntax:** save layer|\$lay formula \$fname

**\$fname:** String with formula filename (.frm) and path.

#### Example

```
save $layer1 formula $form
```

---

**say status**

Updates the status dialog with a message (if string) or updates the percent done gauge to number (between 0 and 100%) or both.

**Syntax:** `say status $number|$string`

`say status $number $string`

**\$number:** Number, representing percentage done

**\$string:** String with status message

**Examples**

```
$percent = 80
say status $percent"Opening image window\n"
say status 100 "Finished"
```

---

**say warning**

Opens an ER Mapper warning dialog box

**Syntax:** `say warning "text"`

**Example**

```
checkdataset:
if ($filename!= "") then goto DatasetOK
  say warning "Please enter a filename"
  goto wizard_page_1
DatasetOK:
#carry on
```

---

**select algorithm**

Changes the current algorithm pointer to point to the specified algorithm.

**Syntax:** `select $alg`

**Examples**

```
select $window_alg
```

---

**select filter**

Sets the current filter to the specified filter.

**Syntax:** `select $fil`

**Example**

```
select $fil
```

---

**select formula**

Sets the current formula to the specified formula.

**Syntax:** `select $for`

**Example**

```
select $for1
```

---

**select layer**

Sets the current layer pointer to point to the specified layer.

**Syntax:** `select $lay`

**Example**

```
select $layer1
```

---

**select surface**

Selects the given surface and makes it the current surface.

**Syntax:** `select $srf`

**Example**

```
select $window_surface
```

---

**select transform**

Sets the current transform to the specified transform.

**Syntax:** `select $tra`

**Example**

```
select $trans1
```

---

**select window**

Updates the current window pointer to point to the given window.

**Syntax:** `select $win`

**Example**

```
select $win
```

---

**set (formula) input to band**

Sets the current formula input to the image band given.

**Syntax:** `set input $n1 to band $n2`

**\$n1 \$n2:** Numbers, representing input and band numbers

**Example**

```
set input 1 to band 1
```

---

### set (page) contents bottomright from topleft

Calculates and sets the bottom right contents extents coordinate, based on the topleft coordinate, page size, borders, and scale. This functionality makes the implementation of map templates much easier because you do not need to change the page layout to accommodate a different image.

**Syntax:** set [\$alg] contents bottomright from topleft

#### Example

```
set $alg contents bottomright from topleft
```

---

### set (page) contents extents to algorithm extents

Set the contents extents to the algorithm extents (where **algorithm** is the current algorithm). This is equivalent to the "Snapshot" button on the Page Setup dialogue.

**Syntax:** set contents extents to [algorithm|\$alg] extents

#### Example

```
set contents extents to $alg extents
```

---

### set (page) contents topleft|bottomright

Sets the page contents extents coordinates to that specified by \$value.

**Syntax:** set [\$alg] contents topleft|bottomright northings|latitude|meters\_y [to] \$value

```
set [$alg] contents topleft|bottomright eastings|longitude|meters_x [to] $value
```

\$value: Number, representing extent value

#### Example

```
set $alg contents topleft eastings to $template_tl_e_extent
```

---

### set algorithm coordsys changeable to (flag)

By default, when a layer is added to an algorithm or the data source of an existing layer in the algorithm is changed, the algorithm's coordinate system will change to the coordinate system of the layer's data source when the coordinate system of the layer's datasource is not the same as the algorithm's coordinate system and not able to be reprojected on the fly to the algorithm's coordinate system.

Calling this command with the flag set to FALSE will prevent the algorithm coordinate system changing when a layer is added to an algorithm or the data source of an existing layer in the algorithm is changed.

**Syntax:** set algorithm coordsys changeable to true|false

#### Example

```
set $alg coordsys changeable to true
set algorithm coordsys changeable to false
```

---

## set page autovary\_value

This can be used to calculate one of the page setup variables based on the Constraints setting. For example:

- When the constraint is "Fixed Page: Extents from Zoom", this command sets the scale to the largest that will fit on the page.
- When the constraint is "Auto Vary: Page", this command sets the page size to whatever is needed to fit in the desired scale and borders.
- When the constraint is "Auto Vary: Borders", this command sets the bottom and/or right border sizes to whatever is needed to accommodate the specified scale and page size.
- When the constraint is "Auto Vary: Scale", this command sets the scale to whatever is needed to create a map with the specified page and border sizes.

**Syntax:** set [\$alg] page autovary\_value

### Example

```
set page autovary_value
```

---

## set algorithm background (color)

Changes the algorithm's background color to the given RGB values or to a value specified by a variable.

**Syntax:** set algorithm|\$alg background to \$red \$green \$blue

**set [algorithm|\$alg] background to colorval \$colorvariable**

**\$red \$green \$blue:** RGB numeric values for the required color.

**\$colorvariable:** Color specification in the form of a string (e.g. "red"), an RGB triple (e.g. 255,0,0 for red), or a variable which has been set to a color specification.

### Examples

```
set algorithm background to 250 245 50
$red = 200
$green = 150
$blue = 30
set $alg background to $red $green $blue
set algorithm background to colorval 250 245 50
set background to color "red"
$background_color = "white"
set $alg background to colorval $background_color
```

---

### set algorithm coordsys

Sets the algorithm's coordinate system type to the given type.

**Syntax:** set algorithm|\$alg coordsys to \$csys|raw|en|ll

**\$csys:** Coordsys type variable: Value can be raw, en or ll

#### Examples

```
$cssystem = en
set algorithm coordsys to $cssystem
set $alg coordsys to en
```

---

### set algorithm datum|projection

Sets the algorithm's datum or projection to the given type. This will cause layers of an incompatible type to be turned off within the algorithm.

**Syntax:** set algorithm|\$alg datum to \$datumname

**set algorithm|\$alg projection to \$projname**

**\$datumname:** String representing required datum

**\$projname:** String representing required projection.

#### Examples

```
$datum = "NAD27"
set algorithm datum to $datum
set algorithm datum to "NAD27"
set $alg projection to "NUTM11"
```

---

### set algorithm description

Changes the current or specified algorithm description to the given text.

**Syntax:** set algorithm|\$alg description to \$text

**\$text:** Algorithm description text string

#### Examples

```
$text = "This algorithm"
set $alg description to $text
set algorithm description to "Pseudocolor Aspect"
```



---

### set algorithm lut (color table)

Changes the pseudocolor LUT for the first surface in the algorithm to the given lut file.

**Syntax:** `set algorithm|$alg lut [to] $lutname`

**\$lutname:** String representing color lookup table filename. It should be within the `ERMAPPER\lut` directory and specified within quotes but without the file extension.

#### Examples

```
$lut = "greyscale"
set $alg lut to $lut
set algorithm lut to "greyscale"
```

---

### set algorithm mode

Changes the processing mode for the first surface in the algorithm to the given mode.

**Syntax:** `set algorithm|$alg mode to $mode|pseudo|rgb|hsi`

**\$mode:** String variable representing required mode, "pseudo", "rgb" or "hsi"

#### Examples

```
set algorithm mode to rgb
$alg_mode = "hsi"
set $alg mode to $alg_mode
```

---

### set algorithm mosaic type

Sets the mosaic type (where different datasets overlay) to the given type. Overlay overwrites earlier layers with latter ones; Feather blends the border.

**Syntax:** `set algorithm|$alg mosaic type to $mtype|overlay|feather`

**\$mtype:** Mosaic type variable: overlay or feather

#### Examples

```
$mtype = feather
set algorithm mosaic type $mtype
set algorithm mosaic type to feather
```

---

### set algorithm supersample type

Sets the current or specified algorithm supersample type.

**Syntax:** `set algorithm|$alg supersample type to $supertype|nearest|bilinear`

**\$supertype:** SuperSample type variable: value can be nearest or bilinear

#### Examples

```
$supertype = bilinear
set algorithm supersample type to $supertype
set $alg supersample type to nearest
```

---

**set algorithm topleft|bottomright (extents)**

Sets the current or specified algorithm topleft and bottomright extent fields in easting/northing, latitude/longitude or raw coordinate systems.

**Syntax:** **set algorithm|\$alg topleft|bottomright eastings|longitude|meters\_x to \$number**

**set algorithm|\$alg topleft|bottomright northings|latitude|meters\_y to \$number**

**\$number:** Number, representing extents value.

**Examples**

```
$tl_east = 475912.476235
set algorithm topleft eastings to $tl_east
set algorithm topleft eastings to 475912.476235
```

---

**set algorithm units|rotation**

Sets the units or rotation to the given value.

**Syntax:** **set algorithm|\$alg units to \$units**

**set algorithm|\$alg rotation to \$value**

**\$units:** String representing units: "meters", "feet", "yards", "links", "chains", "roods", "breaaly units" or "natural"

**\$value:** Numerical value representing rotation in degrees

**Examples**

```
$units = "natural"
set algorithm units to $units
set $alg units to "natural"
$rotate = 0
set $alg rotation to $rotate
set algorithm rotation to 0
```

---

**set filter description**

Changes the current or specified filter description to the given text.

**Syntax:** **set filter|\$fil description to \$text**

**\$text:** Filter description text string

**Examples**

```
$text = "This filter"
set $fill description to $text
set filter description to "High_pass"
```

---

### set filter matrix (element)

Sets an element of the current or specified filter matrix. Valid only on convolution and threshold filters.

**Syntax:** `set filter|$fil matrix $x $y $number`

**\$x \$y:** Numbers, representing coordinates of element in matrix

**\$number:** Number, value to set in element

#### Examples

```
$number = 4
set filter matrix 1 1 $number
set filter matrix 1 1 1
```

---

### set (usercode) filter params

Set the current or specified user filter parameter string. Valid only on usercode filters.

**Syntax:** `set filter|$fil params to $paramstring`

**\$paramstring:** String, with filter parameters

#### Example

```
$params = "$SYS_STDOUT"
set filter params to $params
```

---

### set filter postsampled (flag)

Sets whether the current or specified filter can process resampled data, or source data.

**Syntax:** `set filter|$fil postsampled true|false`

#### Example

```
$set filter postsampled false
```

---

### set filter rows|cols

Sets the number of rows/columns for the current or specified filter to \$number. \$number must be odd.

**Syntax:** `set filter|$fil rows|cols [to] $number`

**\$number:** Number, representing number of rows or columns. Must be odd

#### Example

```
$filrows = 3
set filter rows to $filrows
set $fill cols to 3
```

---

**set filter scale|threshold**

Sets the scale or threshold for the current or specified filter. Valid only on convolution and threshold filters.

**Syntax:** **set filter|\$fil scale|threshold [to] \$value**

**\$value:** Number, representing scale or threshold value

**Example**

```
$value = 1
set filter scale to $value
set filter scale to 1
```

---

**set filter type**

Sets the filter type of the current or specified filter.

**Syntax:** **set filter|\$fil type [to] \$ftype|convolution|threshold|user**

**\$ftype:** Filter type variable: value can be convolution, threshold or user.

**Example**

```
$filtype = threshold
set filter type to $filtype
set $fil1 type to threshold
```

---

**set (usercode) filter userfile (filename)**

Sets the current or specified user filter source file, for a C usercode filter. Valid only on usercode filters.

**Syntax:** **set filter|\$fil userfile [to] \$filename**

**\$filename:** String with path and name of filter source (.c) file

**Example**

```
$fname = "usercode\kernel\c\newfilter.c"
set filter userfile to $fname
set $fil2 userfile to "usercode\kernel\c\newfilter.c"
```

---

**set (usercode) filter userfunc (filename)**

Sets the current or specified user filter function (.dll) name. Valid only on usercode filters.

**Syntax:** **set filter|\$fil userfunc to \$funcname**

**\$funcname:** String with path and name of filter function (.dll) file

**Examples**

```
$funcname = "usercode\kernel\c\win32\newfilter.dll"
set filter userfunc to $funcname
set $fil2 userfunc to "usercode\kernel\c\win32\newfilter.dll"
```

---

## set formula

Sets the current or specified formula (`$formula` is a string).

**Syntax:** `set formula|$for to $formula`

**\$formula:** String, with formula

### Examples

```
$formula = "i1 + i2"
set formula to $formula
set $for1 to "i1 + i2"
```

---

## set layer azimuth|elevation

Sets the sun shade azimuth/elevation to the given value in degrees.

**Syntax:** `set layer|$lay azimuth|elevation to $deg`

**\$deg:** Number, representing azimuth or elevation in degrees.

### Example

```
set layer azimuth to 45
```

---

## set layer color

Sets the current or specified layer color to the given RGB or color values. Valid only on link layers.

**Syntax:** `set layer|$lay color to $red $green $blue`

**set layer color to colorval \$colorvariable**

**\$red \$green \$blue:** RGB numeric values for the required color.

**\$colorvariable:** Color specification in the form of a string (e.g. "red"), an RGB triple (e.g. 255,0,0 for red), or a variable which has been set to a color specification.

### Examples

```
set layer color to 1.0 1.0 1.0
set $lay1 color to 1.0,1.0,1.0
$lay_color = "red"
set layer color to colorval $lay_color
set $layer1 color to colorval "red"
set layer color to colorval 123,155,100
```

---

**set layer dataset (filename)**

Sets the image file name for the current or specified layer to \$dsname.

**Syntax:** set layer|\$lay dataset to \$dsname

**\$dsname:** String specifying the image name (within quotes), including the file extension. The path should be either absolute or relative to the current directory

**Example**

```
$dsname = "examples\Shared_Data\testlayer.ers"  
set $layer1 dataset to $dsname
```

---

**set layer description**

Changes the layer description to the text given.

**Syntax:** set layer|\$lay description to \$text

**\$text:** String with layer description

**Examples**

```
$lay_desc = "First red"  
set layer description to $lay_desc  
set $layer1 description to "Second red"
```

---

**set layer editable (flag)**

Sets the editable flag for current or specified layer. Valid only for link layers.

**Syntax:** set layer|\$lay editable [to] true|false

**Example**

```
set layer editable to true
```

---

**set layer edit|init program (name)**

Sets the current or specified layer's edit/init program to the given name. Valid only on link layers.

**Syntax:** set layer|\$lay edit|init program to \$fname

**\$fname:** String, with program name.

**Examples**

```
set layer edit program to "erm_link"  
set layer init program to "erminit_oracle"
```

---

## set layer formula

Sets the current or specified layer formula.

**Syntax:** `set layer|$lay formula to $formula`

**\$formula:** String, with formula.

### Examples

```
$form = "i1 + i2"
set layer formula to $form
set layer formula to "i1 + i2"
```

---

## set layer input to band

Sets the designated layer input to the specified band number

**Syntax:** `set layer|$lay input $n1 to band $n2`

**\$n1 \$n2:** Number

### Examples

```
set layer input 1 to band 1
set layer $input_no to band $band_no
```

---

## set layer link extension

Sets the link file extension of the current or specified layer to the given string.

**Syntax:** `set layer link extension to $extent`

**\$extent:** String with link extension. This field can take three forms:

"": no extension.

".erv": a normal file extension.

"erv\_chooser": a program to generate a list of choices from which the user must choose one.

### Example

```
set layer link extension to ".erv"
```

---

## set layer link type to monocolour|truecolour

Sets the link type of the current or specified layer to monocolour or truecolour. Valid only on link layers. Note the English spelling of 'colour'.

**Syntax:** `set layer|$lay link type to monocolour|truecolour`

### Example

```
set layer link type to truecolour
```

---

### set layer shading on|off

Turns sunshading on/off for the current or specified layer.

**Syntax:** set layer|\$lay shading on|off

#### Example

```
set layer shading on
```

---

### set layer type

Changes the current or specified layer to the given type.

**Syntax:** set layer|\$lay type to \$stype|pseudocolor|red|green|blue|hue|saturation|intensity|height|classification|classdisplay|link

**\$stype:** Layer type variable: see [“Variables”](#) for allowed values.

#### Examples

```
$lay_type = hue
set layer type to $lay_type
set $layer1 type to green
```

---

### set page center horizontal|vertical

Centers the contents of the page horizontally or vertically.

**Syntax:** set [\$alg] page center horizontal|vertical

#### Example

```
set $alg page center horizontal
```

---

### set page constraints

Sets the page constraints of the current or specified algorithm.

**Syntax:** set [\$alg] page constraints to \$constr|zoom|page|border|scale

**\$constr:** Constraints variable: value can be zoom, page, border or scale

#### Examples

```
$page_constraint = border
set $alg page constraints to $page_constraint
set page constraints to scale
```

---

### set page extents from contents

Calculates and sets the page extents of the algorithm based on the contents extents, the borders, and the scale.

**Syntax:** set [\$alg] page extents from contents

#### Example

```
set $alg page extents from contents
```



---

## set page scale

Sets the page scale to a specific number or the maximum scale possible.

**Syntax:** `set [$alg] page scale to $scale|max`

**\$scale:** Number, representing scale value

### Examples

```
$page_scale = 1000
set $alg page scale to $page_scale
set $alg page scale to max
set page scale to 1000
```

---

## set page size

Sets the page size of the current or specified algorithm.

**Syntax:** `set [$alg] page size to $size`

**\$size:** String with standard page size: value can be one of the following:

- "US Letter", "US Letter - landscape",
- "US Legal", "US Legal - landscape",
- "A0", "A0 - landscape",
- "A1", "A1 - landscape",
- "A2", "A2 - landscape",
- "A3", "A3 - landscape",
- "A4", "A4 - landscape",
- "A5", "A5 - landscape",
- "A6", "A6 - landscape",
- "B3", "B3 - landscape",
- "B4", "B4 - landscape",
- "B5" or "B5 - landscape"

### Examples

```
$page_size = "US Letter"
set $alg page size to $page_size
set page size to "US Letter"
```

---

**set page topleft|bottomright (extents)**

Sets the page extents coordinates to that specified in the variable \$value.

**Syntax:** **set [\$alg] page topleft|bottomright northings|latitude|meters\_y [to] \$value**

set [\$alg] page topleft|bottomright eastings|longitude| meters\_x [to] \$value

\$value: Number, representing extent value

**Example**

```
set $alg page topleft eastings to $tl_e_extent
```

---

**set page top|bottom|left|right border**

Sets the page borders.

**Syntax:** **set [\$alg] page top|bottom|left|right border to \$number mm|inches**

**\$number:** Number, representing size in inches or millimeters

**Examples**

```
$page_top = 1.0
set $alg page top border to $page_top inches
set page top border to 1.0 inches
```

---

**set (algorithm) page view mode**

Sets the page view mode of the current or specified algorithm to normal or layout.

**Syntax:** **set [\$alg] page view mode to \$pvmode|normal|layout**

**\$pvmode:** Page view mode variable: value can be normal or layout

**Example**

```
set $alg page view mode to layout
```

---

**set page width|height**

Sets the specified page size parameter of the current or specified algorithm.

**Syntax:** **set [\$alg] page width|height [to] \$number inches|mm**

**\$number:** Number, representing size in inches or millimeters

**Examples**

```
$page_width = 5
set $alg page width to $page_width inches
set $alg page width to 5 inches
```

---

## set pointer mode

Sets the mouse pointer to the specified mode: zoom, zoombox, roam(hand) or pointer.

**Syntax:** `set pointer mode to zoom|zoombox|roam|pointer`

### Examples

```
set pointer mode to zoom
set pointer mode to zoombox
set pointer mode to roam
```

---

## set preference

Adds or updates the preference 'name' entry in the user's preference file as a string or number.

**Syntax:** `set preference "name" [to] $string|$number[boolean|integer|double]`

**"name":** String with name of entry in preference file. The recommended format is "Class:Name:Variable".

**\$string:** String with value for entry

**\$number:** Number with value for entry

### Examples

```
set preference "Wizard:Image:MainChoice" $page_main_choice
set preference "MACHINECONFIG_HAS-RUN" TRUE boolean
```

---

## set preference (color)

Sets the color preference entry in the user's preference file.

**Syntax:** `set preference "colorname" to $red $green $blue`

**set preference "colorname" to colorval \$color|\$red,\$green,\$blue**

**"colorname":** Color preference entry name

**\$red \$green \$blue:** RGB numeric values for the required color.

**\$color:** Color specification in the form of a string (e.g. "red"), an RGB triple (e.g. 255,0,0 for red), or a variable which has been set to a color specification.

### Examples

```
set preference "my_color_pref" to 155 203 200
set preference "my_color_pref" to 155,203,200
set preference "my_color_pref" to colorval $new_color
set preference "my_color_pref" to colorval "green"
set preference "my_color_pref" to 155,203,200
```

---

### set surface description

Changes the current or specified surface description to the given text.

**Syntax:** `set surface|$srf description to $text`

**\$text:** Surface description text string

#### Examples

```
$text = "This surface"
set $srf1 description to $text
set surface description to "Main surface"
```

---

### set surface lut (color table)

Sets the Color Table for the current surface.

**Syntax:** `set surface lut [to] $lut`

**\$lut:** String representing color lookup table filename. It should be within the %ERMAPPER%\lut directory and specified within quotes but without the file extension.

#### Examples

```
set surface lut to $new_lut
set surface lut to "pseudocolor"
```

---

### set surface mode (color)

Sets the color mode for the current surface.

**Syntax:** `set surface mode [to] $surface_mode|rgb|hsi|pseudo`

**\$surface\_mode:** Mode type variable: value can be rgb, hsi or pseudocolor (or pseudo)

#### Examples

```
$new_mode = rgb
set surface mode to $new_mode
set surface mode to pseudo
```

---

### set surface name

Sets the current surface name to the given name.

**Syntax:** `set surface name [to] $string`

**\$string:** String with surface name.

#### Examples

```
$surface_name = "Top surface"
set surface name to $surface_name
set surface name to "Bottom surface"
```

---

**set surface zscale|zoffset|transparency**

Sets the Z Scale, Z Offset or transparency of the current surface to \$value.

**Syntax:** set surface zscale|zoffset|transparency [to] \$value

**\$value:** Number representing required value.

**Example**

```
set surface transparency to 20
```

---

**set transform clip**

Applies a clip of \$pct percent to the current or specified transform. If \$pct is not specified 99.0 is used.

**Syntax:** set transform|\$tra clip [to \$pct]

**\$pct:** Number, representing percentage clip

**Example**

```
set transform clip to 90
```

---

**set transform input|output min|max (limits)**

Sets the current or specified transform input/output limits.

**Syntax:** set transform|\$tra input|output min|max to \$number

**\$number:** Number, representing input/output limits

**Examples**

```
$inmax = 361
set transform input max to $inmax
set transform input max to 7
```

---

**set transform limits (actual or percentage)**

Sets the current or specified transform limits to \$percent or actual.

**Syntax:** set transform|\$tra limits to actual|\$percent

**\$percent:** Number, representing percentage

**Example**

```
set transform limits to actual
set transform limits to 50
```

---

### set transform to gaussian equalize

Applies a gaussian equalize operation to the current or specified transform. If the gaussian envelope halfwidth is not specified a halfwidth of 3 standard deviations is used.

**Syntax:** `set transform|$tra to gaussian equalize [$halfwidth]`

**\$halfwidth:** Number, representing envelope halfwidth in standard deviations

#### Example

```
set transform to gaussian equalize
```

---

### set transform to histogram equalize

Applies a histogram equalize operation to the current or specified transform.

**Syntax:** `set transform|$tra to histogram equalize`

#### Example

```
set transform to histogram equalize
```

---

### set transform type

Sets the current or specified transform type.

**Syntax:** `set transform|$tra type to $ttype|linear|exp|log|hist`

**\$ttype:** Transform type variable: value can be linear, exp, log or hist

#### Example

```
set transform type to linear
```

---

### set view mode

Sets the view mode of the current or specified algorithm to 2D, 3D perspective or 3D flythru.

**Syntax:** `set [$alg] view mode to $vmode|2d|perspective|flythru`

**\$vmode:** ViewMode variable: value can be 2d, perspective or flythru

#### Example

```
set $alg view mode to perspective
```

---

### set window geolink mode

Various geolink functionality.

**Syntax:** `set window|$win geolink mode to none|window|screen|overview`

#### Example

```
set window geolink mode to window
```

---

## setenv (environment variable)

Set the given environment.



---

*This changes the current environment within ER Mapper.*

**Syntax:** `setenv $envname $value`

**\$envname:** String with name of environment variable

**\$value:** String with value to set the environment variable to

### Example

```
setenv "ERM_MACHINE_TYPE" "win32"
```

---

## show image (in container)

Displays the specified image in the container.

**Syntax:** `show image $filename`

**\$filename:** String with path and name of image file. path name can be absolute or relative to the %ERMAPPER%\icons directory

### Examples

```
$image = "standard_icons\image_main"  
show image $image  
show image "standard_icons\image_main"
```

---

## split string

Split the given string into an array of substrings at the given token. The array is terminated by an empty string. Specifying empty double quotes as the split character splits a string into an array of single letter elements.

```
$string = "first second third"
$array = split $string at " "
$array[1] -> "first"
$array[2] -> "second"
$array[3] -> "third"
$array[4] -> "" - terminator
$array[5] -> invalid
```



---

*The first array index will always be a 1.*

### Syntax: split \$string at \$token

**\$string:** String to be split

**\$token:** String with character where string must be split

### Example

```
$filenames = split $string at "\n"
```

---

## surface active

Checks whether the current surface is active, and returns TRUE (1) for active, and FALSE (0) for inactive.

### Syntax: surface active

### Examples

```
$surface_is_on = surface active
```

---

## system command

Executes a system command from within the script.

### Syntax: system \$command [status ["Title"]]

**\$command:** System command to be executed "&" at the end of the command will make it run in the background.

**status:** Optional status dialog box

**"Title":** String with status box title (optional)

### Examples

```
$cmd = "ermbalance calcclip" + $balance_file_spec + "&"
system $cmd
system "ermbalance calcclip" + $balance_file_spec + "&"
```



---

**transform2 = transform1**

Assign specified transform (e.g. \$tra1) to another variable (e.g. \$tra2).



---

*\$tra2 and \$tra1 refer to the same object, so deleting \$tra1 will invalidate \$tra2. Use the copy or duplicate transform command to create a separate object.*

**Syntax: \$tra2 = \$tra1**

**Example**

```
$tra2 = $tra1
```

---

**turn layer on|off**

Turns the current or specified layer on or off.

**Syntax: turn layer|\$lay on|off**

**Example**

```
turn $layer1 on
```

---

**turn surface on|off**

Enables/Disables the current surface.

**Syntax: turn surface on|off**

**Example**

```
turn surface on
```

---

**window2 = window1**

Copies the window reference from win1 to win2, that is, both \$win1 and \$win2 point to the same object.

**Syntax: \$win2 = \$win1**

---

**Wizard close**

The "Wizard Close" command is needed because you often *can't* automatically close a wizard when the user presses the **Finish** button. You usually need to check the necessary parameters have been entered. In this case you would check everything is fine, and if it is, close the wizard using the "Wizard Close" command, and if it is not, go back to the appropriate page of the wizard. You can use Exit to exit a script and close down the wizard automatically but this is not recommended.

**Syntax: Wizard close [name]**

**name:** If you don't specify a Name, all wizards are closed. If you specify a Name, then the named wizard, and all sibling wizards created after it, are closed.

---

## WizardPage begin|end

This marks the beginning and end of a new wizard page. 'WizardName' is the name of the wizard within the script. If it is a different name from the previous Wizard Block a new wizard window is drawn; if it is the same as the name in the previous Wizard Block, the previous window is redrawn with the new contents. WizardName is optional for second and subsequent pages: if omitted it defaults to the name from the previous Wizard Block.

**Syntax: WizardPage begin "WizardName" title "title\_text"  
..... (container blocks)  
WizardPage end**

**title "title\_text":** Specifies the title which will appear on the Title Bar of the wizard page on screen. This should include the name of the Wizard (which indicates what it is used for), followed by a hyphen and the step number. For example,  
title "ClassificationWizard - Step 1 of 4"

**container blocks:** If no container block is defined a default container block the size of the whole page will be included.

**WizardPage end:** Defines the end of the wizard page.

### Example

```
WizardPage begin "WizardName"#title for first page
  title "Page1Name" #if a new wizard
  container begin "Image"
    container information           #usually image info
    ...                             #on first page
  container end
  container begin "DataEntry"
    container information
    ...
  container end
  container begin "PageControls" #control buttons
    ask action "< Back"
    ask action "Next >" goto label1
    ask action "Cancel" close goto label2
  container end
  wizard close
WizardPage end
```

---

## zoom

Zoom in or out using specified option.

**Syntax: previous zoom**

**zoom in|out**

**zoom to all datasets|current datasets|all raster datasets|all vector datasets|contents extents|page extents|page contents**

### Examples

```
previous zoom
zoom in
set pointer mode to roam
zoom to all datasets
zoom to page contents
```

# Sensor Platform Characteristics

This section provides details of various sensor platforms whose data are supported by ER Mapper.

## NOAA

- SHORT NAME: NOAA\_AVHRR
- PLATFORM: NOAA
- SENSOR: AVHRR(Advanced Very High Resolution Radiometer)
- TYPE: Polar orbiting satellite.
- DESCRIPTION: Designed to provide information for hydrologic, oceanographic and meteorologic studies, although data provided by the sensor does find application also to solid earth monitoring.
- ALTITUDE: 700 - 1500 km
- REPEAT CYCLE: 2 per day from two satellites
- GROUND SWATH: 2700 km
- NADIR RESOLUTION: 1.1 km
- DYNAMIC RANGE: 10 bit
- SPECTRAL BANDS:

Band No.	Wavelength ( $\mu$ ) microns	Bandwidth ( $\mu$ ) microns	Central Wavelength ( $\mu$ ) microns
1	0.58-0.68	0.1	0.63
2	0.725-1.1	0.375	0.9125
3	3.55-3.93	0.38	3.74
4	10.3-11.3	1.0	10.8
5	11.5- 12.5	1.0	12.0

ERMAPPER SENSORTYPE FILE:  
**cameras\_and\_sensors\AVHRR.ers**

**AVHRR.ers** file is as follows:

```
DatasetHeader Begin
    Version = "1.0"
    Name = "NOAA (AVHRR) satellite"
    DataSetType = ERStorage
    DataType = Raster
    RasterInfo Begin
```

```

CellType = Unsigned8BitInteger
NullCellValue = 0
CellInfo Begin
    Xdimension = 1100
    Ydimension = 1100
    Xoverlap = 0
    Yoverlap = 0
CellInfo End
NrOfBands= 5
BandId Begin
    Value = "0.63"
    Width = 0.1
    Units = "um"
BandId End
BandId Begin
    Value = "0.9125"
    Width = 0.375
    Units = "um"
BandId End
BandId Begin
    Value = "3.74"
    Width = 0.38
    Units = "um"
BandId End
BandId Begin
    Value = "10.8"
    Width = 1.0
    Units = "um"
BandId End
BandId Begin
    Value = "12.0"
    Width = 1.0
    Units = "um"
BandId End
RasterInfo End
DatasetHeader End

```

## Geoscan MSS Mark 2

- SHORT NAME: GEOSCAN\_2
- PLATFORM: Airborne scanner
- SENSOR: Multi-spectral
- DESCRIPTION: Has roll, pitch and yaw stabilisation; can record UTM co-ordinates for each pixel; 6 calibration sources on scanner; records image data onto optical disk media. Up to 10 bisible/NIR channels selectable from 32 standard detector intervals between 480-1000 nm, plus 8 MIR channels of equal width between 2.0 - 2.5 um, plus 6 TIR channels of equal width between 8.0 - 12.0 um.

- MAX FOV(DEGREES): 92.16
- MAX IFOV(DEGREES): 3 x 2.25
- DYNAMIC RANGE: 8 or 12 bit (plus 8 bit gain and 8 bit offset)
- SPECTRAL BANDS: Total of 24 bands. Typical configuration, below.

Band No.	Wavelength (μ) microns	Bandwidth (μ) microns	Central Wavelength (μ) microns
<b>VNIR</b>			
1		0.042	0.522
2	0.550-0.617	0.067	0.583
3		0.071	0.645
4	0.681-0.705	0.024	0.693
5	0.705-0.729	0.024	0.717
6	0.729-0.752	0.023	0.740
7	0.819-0.841	0.022	0.830
8	0.862-0.883	0.022	0.873
9	0.904-0.926	0.021	0.915
10	0.945-0.965	0.020	0.955
<b>SWIR</b>			
11	1.820-2.260	0.044	2.044
12	1.868-2.308	0.044	2.088
13	1.912-2.352	0.044	2.136
14	1.956-2.396	0.044	2.176
15	2.000-2.440	0.044	2.220
16	2.044-2.484	0.044	2.264
17	2.088-2.528	0.044	2.308
18	2.132-2.572	0.044	2.352
<b>TIR</b>			
19		0.530	8.64
20		0.530	9.17
21		0.530	9.70
22		0.530	10.22
23		0.530	10.75
24		0.530	11.28

ERMAPPER SENSORTYPE FILE:  
**cameras\_and\_sensors\Geoscan.ers**  
**Geoscan2.ers** file is shown below.

```

DatasetHeader Begin
    Version = "1.0"
    Name = "Geoscan Mark II 24 band scanner"
    DataSetType = ERStorage
    DataType = Raster

```

```
CoordinateSpace Begin
  ProjectionType = RAW
  CoordinateSystem = RAW
CoordinateSpace End
RasterInfo Begin
  CellType = Unsigned8BitInteger
  NullCellValue = 0
  CellInfo Begin
    Xdimension = 3
    Ydimension = 3
    Xoverlap = 0
    Yoverlap = 0
  CellInfo End
  NrOfCellsPerLine = 1024
  NrOfBands = 24
  BandId Begin
    Value = "0.522"
    Width = 0.042
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.583"
    Width = 0.067
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.645"
    Width = 0.071
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.693"
    Width = 0.024
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.717"
    Width = 0.024
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.740"
    Width = 0.023
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.830"
    Width = 0.022
    Units = "um"
  BandId End
```

```
BandId Begin
  Value = "0.873"
  Width = 0.022
  Units = "um"
BandId End
BandId Begin
  Value = "0.915"
  Width = 0.021
  Units = "um"
BandId End
BandId Begin
  Value = "0.955"
  Width = 0.020
  Units = "um"
BandId End
BandId Begin
  Value = "2.044"
  Width = 0.044
  Units = "um"
BandId End
BandId Begin
  Value = "2.088"
  Width = 0.044
  Units = "um"
BandId End
BandId Begin
  Value = "2.136"
  Width = 0.044
  Units = "um"
BandId End
BandId Begin
  Value = "2.176"
  Width = 0.044
  Units = "um"
BandId End
BandId Begin
  Value = "2.220"
  Width = 0.044
  Units = "um"
BandId End
BandId Begin
  Value = "2.264"
  Width = 0.044
  Units = "um"
BandId End
BandId Begin
  Value = "2.308"
  Width = 0.044
  Units = "um"
BandId End
```

```

BandId Begin
  Value = "2.352"
  Width = 0.044
  Units = "um"
BandId End
BandId Begin
  Value = "8.64"
  Width = 0.530
  Units = "um"
BandId End
BandId Begin
  Value = "9.17"
  Width = 0.530
  Units = "um"
BandId End
BandId Begin
  Value = "9.70"
  Width = 0.530
  Units = "um"
BandId End
BandId Begin
  Value = "10.22"
  Width = 0.533
  Units = "um"
BandId End
BandId Begin
  Value = "10.75"
  Width = 0.533
  Units = "um"
BandId End
BandId Begin
  Value = "11.28"
  Width = 0.533
  Units = "um"
BandId End
RasterInfo End
DatasetHeader End

```

## IRS-IC

- SHORT NAME: IRS-IC (Indian Remote Sensing Satellite)
- PLATFORM: IRS-IC satellite
- SENSOR: There are three sensors, viz. Panchromatic Camera (PAN), Linear Imaging and Self Scanning Sensor (LISS-III) and Wide Field Sensor (WiFS)
- TYPE: Three axis body stabilized satellite.



- **DESCRIPTION:** Provides a systematic and repetitive acquisition of data of the Earth's surface under nearly constant illumination conditions.
- **ORBIT:** Near polar, sun synchronous; nominal 10:30 am descending equatorial crossing.
- **ALTITUDE:** 817 km
- **PERIOD:** 101.35 min
- **REPEAT CYCLE:** 14.2 orbits per day over 24 days (341 revolutions)
- **GROUND SWATH:** The three sensors collect data with different swaths:
  - PAN: 70 km
  - WiFS: 148 km
  - LISS-III: 141 km (visible bands) 148 km (SWIR band)
- **RESOLUTION:** PAN: 5.8 m
  - WiFS: 188.3 m
  - LISS-III: 23.5 m (visible bands) 70.5 m (SWIR band)
- **SPECTRAL BANDS:**

<b>Band No.</b>	<b>Wavelength (<math>\mu</math>) microns</b>	<b>Bandwidth (<math>\mu</math>) microns</b>	<b>Central Wavelength (<math>\mu</math>) microns</b>
<b>PAN</b>			
1	0.5-0.75	0.15	0.575
<b>LISS-III</b>			
1	0.52-0.59 (green)	0.07	0.555
2	0.62-0.68 (red)	0.04	0.64
3	0.77-0.86 (NIR)	0.09	0.815
4	1.55-1.7 (SWIR)	0.15	1.625
<b>WiFS</b>			
1	0.62-0.68 (red)	0.04	0.64
2	0.77-0.86 (NIR)	0.09	0.815

- **ERMAPPER SENSORTYPE FILE: Three sensortype files:**
  - cameras\_and\_sensors\IRS-1C\_Pan.ers
  - cameras\_and\_sensors\IRS-1C\_WiFS.ers

- cameras\_and\_sensors\IRS-1C\_LISS-3.ers

**cameras\_and\_sensors\IRS-1C\_Pan.ers** is shown below.

```

DatasetHeader Begin
  Version = "1.0"
  Name = "IRS-1C Panchromatic Sensor"
  DataSetType = ERStorage
  DataType = Raster
  RasterInfo Begin
    CellType = Unsigned8BitInteger
    NullCellValue = 0
    CellInfo Begin
      Xdimension = 5.8
      Ydimension = 5.8
    CellInfo End
    NrOfBands= 1
    BandId Begin
      Value = "0.625"
      Width = 0.125
      Units = "um"
    BandId End
  RasterInfo End
DatasetHeader End

```

**cameras\_and\_sensors\IRS-1C\_WiFS.ers** is shown below.

```

DatasetHeader Begin
  Version = "1.0"
  Name = "IRS-1C WiFS sensor"
  DataSetType = ERStorage
  DataType = Raster
  RasterInfo Begin
    CellType = Unsigned8BitInteger
    NullCellValue = 0
    CellInfo Begin
      Xdimension = 188.3
      Ydimension = 188.3
    CellInfo End
    NrOfBands= 2
    BandId Begin
      Value = "0.65"
      Width = 0.06
      Units = "um"
    BandId End
    BandId Begin
      Value = "0.815"
      Width = 0.09
      Units = "um"
    BandId End
  RasterInfo End

```

DatasetHeader End

**cameras\_and\_sensorsIRS-1C\_LISS-3.ers** is shown below.

DatasetHeader Begin

```
Version = "1.0"
Name = "IRS-1C LISS-3 sensor"
DataSetType = ERStorage
DataType = Raster
RasterInfo Begin
  CellType = Unsigned8BitInteger
  NullCellValue = 0
  CellInfo Begin
    Xdimension = 23.5
    Ydimension = 23.5
  CellInfo End
  NrOfBands= 4
  BandId Begin
    Value = "0.555"
    Width = 0.07
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.65"
    Width = 0.06
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.815"
    Width = 0.09
    Units = "um"
  BandId End
  BandId Begin
    Value = "1.625"
    Width = 0.075
    Units = "um"
  BandId End
RasterInfo End
```

DatasetHeader End

## **LANDSAT 4 or LANDSAT MSS**

- SHORT NAME: Landsat\_4 or Landsat\_MSS
- PLATFORM: Landsat
- SENSOR: Multispectral Scanner (MSS) & Thematic Mapper (TM)
- TYPE: Second generation earth resources satellite system
- DESCRIPTION:Spacecraft at lower altitude than first generation, to assist in achieving higher resolution and to aid shuttle recovery.

- ORBIT:Near polar, sun synchronous; nominal 9:30 am descending equatorial crossing.
- ALTITUDE:705 km
- PERIOD:98.9 min
- REPEAT CYCLE:14.56 orbits per day over 16 days (233 revolutions)
- GROUND SWATH:185 m
- IFOV81.5 m
- NADIR RESOLUTION:56 x 56 m (MSS)
- DYNAMIC RANGE:7 bits (MSS), 6 bits ( MSS 0.8 - 1.1 um), 8 bits (TM)
- SPECTRAL BANDS:

Band No.	Wavelength (μ) microns	Bandwidth (μ) microns	Central Wavelength (μ) microns
<b>MSS</b>			
1	0.5-0.6	0.1	0.55
2	0.6-0.7	0.1	0.65
3	0.7-0.8	0.1	0.75
4	0.8-1.1	0.3	0.95
<b>TM</b>			
1	0.45-0.52	0.07	0.485
2	0.52-0.60	0.08	0.56
3	0.63-0.69	0.06	0.66
4	0.76-0.90	0.14	0.83
5	1.55-1.75	0.2	1.65
7	2.08-2.35	0.27	2.215
6	10.4-12.5	2.1	11.45

- ERMAPPER SENSORTYPE FILE:Two sensortype files:
  - cameras\_and\_sensors\Landsat\_4.ers
  - and **cameras\_and\_sensors\Landsat\_MSS.ers.**

These files are equivalent, they have been so named for convenience.

**cameras\_and\_sensors\Landsat\_4.ers** is shown below.

```

DatasetHeader Begin
    Version = "1.0"
    Name = "Landsat 4 (MSS) satellite"

```

```

DataSetType = ERStorage
DataType = Raster
RasterInfo Begin
  CellType = Unsigned8BitInteger
  NullCellValue= 0
  CellInfo Begin
    Xdimension = 81.5
    Ydimension = 81.5
    Xoverlap = 0
    Yoverlap = 0
  CellInfo End
  NrOfBands= 4
  BandId Begin
    Value = "0.55"
    Width = 0.1
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.65"
    Width = 0.1
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.75"
    Width = 0.1
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.95"
    Width = 0.3
    Units = "um"
  BandId End
  RasterInfo End
DatasetHeader End

```

## LANDSAT 5 or LANDSAT TM

- SHORT NAME: Landsat\_5 or Landsat\_TM
- PLATFORM: Landsat
- SENSOR: Multispectral Scanner (MSS) & Thematic Mapper (TM)
- TYPE: Second generation earth resources satellite system
- DESCRIPTION: Spacecraft at lower altitude than first generation, to assist in achieving higher resolution and to aid shuttle recovery.
- ORBIT: Near polar, sun synchronous; nominal 9:30 am descending equatorial crossing.

- ALTITUDE: 705 km
- PERIOD: 98.9 min
- REPEAT CYCLE:14.56 orbits per day over 16 days (233 revolutions)
- NADIR X OVERLAP:
- NADIR Y OVERLAP:
- GROUND SWATH: 185 m
- IFOV: 82.5 m
- NADIR RESOLUTION:30 x 30 m (TM), 120 x 120 m (TM 104. - 12.5 um)
- DYNAMIC RANGE:7 bits (MSS), 6 bits ( MSS 0.8 - 1.1 um), 8 bits (TM)
- SPECTRAL BANDS:

Band No.	Wavelength (μ) microns	Bandwidth (μ) microns	Central Wavelength (μ) microns
<b>MSS</b>			
1	0.5-0.6	0.1	0.55
2	0.6-0.7	0.1	0.65
3	0.7-0.8	0.1	0.75
4	0.8-1.1	0.3	0.95
<b>TM</b>			
1	0.45-0.52	0.07	0.485
2	0.52-0.60	0.08	0.56
3	0.63-0.69	0.06	0.66
4	0.76-0.90	0.14	0.83
5	1.55-1.75	0.2	1.65
7	2.08-2.35	0.27	2.215
6	10.4-12.5	2.1	1

- ERMAPPER SENSORTYPE FILE: Two sensortype files:
  - **cameras\_and\_sensors\Landsat\_5.ers** and
  - **cameras\_and\_sensors\ Landsat\_TM.ers.**

These files are equivalent, they have been so named for convenience. **cameras\_and\_sensors\Landsat\_5.ers** is shown below:

```
DatasetHeader Begin
    Version = "1.0"
```

```

Name = "Landsat 5 (TM) satellite"
DataSetType = ERStorage
DataType = Raster
RasterInfo Begin
  CellType = Unsigned8BitInteger
  NullCellValue= 0
  CellInfo Begin
    Xdimension = 30.0
    Ydimension = 30.0
    Xoverlap = 0
    Yoverlap = 0
  CellInfo End
  NrOfBands= 7
  BandId Begin
    Value = "0.485"
    Width = 0.07
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.56"
    Width = 0.08
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.66"
    Width = 0.06
    Units = "um"
  BandId End
  BandId Begin
    Value = "0.83"
    Width = 0.14
    Units = "um"
  BandId End
  BandId Begin
    Value = "1.65"
    Width = 0.2
    Units = "um"
  BandId End
  BandId Begin
    Value = "11.45"
    Width = 2.1
    Units = "um"
  BandId End
  BandId Begin
    Value = "2.215"
    Width = 0.27
    Units = "um"
  BandId End
  RasterInfo End
DatasetHeader End

```

## LANDSAT 7

- SHORT NAME: Landsat\_7
- PLATFORM: Landsat
- SENSOR: Enhanced Thematic Mapper (ETM+)
- TYPE: Third generation earth resources satellite system
- ORBIT: Near polar, sun synchronous; nominal 10:00 am descending equatorial crossing.
- ALTITUDE: 705 km
- PERIOD: 98.884 min
- REPEAT CYCLE: 14.56 orbits per day over 16 days (233 revolutions)
- GROUND SWATH: 185 Km
- IFOV 170Km
- NADIR RESOLUTION: 30 x 30 m (TM), 120 x 120 m (TM 104. - 12.5 um)
- DYNAMIC RANGE: 8 bits (ETM+)
- SPECTRAL BANDS:

Band Number	Spectral Range(microns)	Ground Resolution(m)
1	0.45 to 0.515	30
2	0.525 to 0.605	30
3	0.63 to 0.690	30
4	0.75 to 0.90	30
5	1.55 to 1.75	30
6	10.40 to 12.5	60
7	2.09 to 2.35	30
Pan	0.52 to 0.90	15

## SPOT

- SHORT NAME: SPOT\_HRV
- PLATFORM: SPOT
- SENSOR: High Resolution Visible (HRV).



- SPOT 4 has two HRV sensors and an additional very wide angle vegetation instrument.
- TYPE: Earth Resource Satellite System
- DESCRIPTION: The SPOT HRV instruments consist of a linear array of charge coupled device (CCD) detectors. Each detector in the array scans a strip in the along track direction - this reduces the need for mechanical scanning and a wide swath can be imaged. A higher spatial resolution is also possible due to the long effective dwell time.

The vegetation instrument in SPOT 4 uses the same spectral bands as the HRV-IR instruments (B2, B3 and mid-IR) plus an additional band known as B0 (0.43-0.47  $\mu\text{m}$ ) for oceanographic applications

- ORBIT: Sun synchronous, 98.7 degrees inclination 10:30 am nominal equator crossing.
- ALTITUDE: 832 km
- REPEAT CYCLE: 26 days
- NADIR X OVERLAP:
- NADIR Y OVERLAP:
- GROUND SWATH: 117 km (Richards) 185 km (Harrison)
- NADIR RESOLUTION: 20 x 20 m (MSS), 10 x 10 m (Panchromatic).
- In SPOT 4 the Panchromatic band has been replaced by having (MSS) band 2 also operate with a 10 x 10 m resolution.
- DYNAMIC RANGE: 8 bits
- SPECTRAL BANDS:

Band No.	Wavelength ( $\mu$ ) microns	Bandwidth ( $\mu$ ) microns	Central Wavelength ( $\mu$ ) microns
<b>MSS</b>			
1	0.50-0.59	0.09	0.545
2	0.61-0.68	0.07	0.645
3	0.79-0.89	0.1	0.84
4 (SPOT 4)	1.58-1.75	0.17	1.665
<b>Panchromatic (not in SPOT 4)</b>			
	0.51-0.73	0.22	0.62

- ERMAPPER SENSORTYPE FILE: Two sensortype files:

- **cameras\_and\_sensors\SPOT\_XS.ers** and
- **cameras\_and\_sensors\SPOT\_PAN.ers**.

**cameras\_and\_sensors\SPOT\_PAN.ers** is shown below:

```

DatasetHeader Begin
  Version = "1.0"
  Name = "Spot 1 panchromatic"
  DataSetType = ERStorage
  DataType = Raster
  RasterInfo Begin
    CellType = Unsigned8BitInteger
    NullCellValue = 0
    CellInfo Begin
      Xdimension = 10.0
      Ydimension = 10.0
    CellInfo End
    NrOfBands= 1
    BandId Begin
      Value = "0.65"
      Width = 0.16
      Units = "um"
    BandId End
  RasterInfo End
DatasetHeader End

```

The **cameras\_and\_sensors\SPOT\_XS.ers** is shown below

```

DatasetHeader Begin
  Version = "1.0"
  Name = "Spot 1 multispectral"
  DataSetType = ERStorage
  DataType= Raster
  RasterInfo Begin
    CellType = Unsigned8BitInteger
    NullCellValue = 0
    CellInfo Begin
      Xdimension = 20.0
      Ydimension = 20.0
    CellInfo End
    NrOfBands= 3
    BandId Begin
      Value = "0.545"
      Width = 0.09
      Units = "um"
    BandId End
    BandId Begin
      Value = "0.645"
      Width = 0.07
      Units = "um"
    BandId End
  RasterInfo End
DatasetHeader End

```

```
BandId Begin
  Value = "0.84"
  Width = 0.1
  Units = "um"
BandId End
RasterInfo End
DatasetHeader End
```



# Data suppliers

Sample images include those obtained by Earth Resource Mapping for development and testing purposes as well as for general distribution with the ER Mapper software.

Earth Resource Mapping, the developers of ER Mapper, do not provide the service of a bureau for remote sensed and geophysical data. In order to promote both the flexibility of ER Mapper and the images available, we approached the image processing community and suppliers of remote sensed and geophysical data to provide a sample of their data for testing purposes and/or as part of the distributed product.

The sources of images shown below is by no means exhaustive, but is complete in the sense that remote sensed and geophysical images are commercially available from the organizations listed.

## **Australian Centre for Remote Sensing (ACRES)**

The Australian Centre for Remote Sensing provides Landsat TM and MSS data products and has established a series of ACRES distributors throughout Australia. Each distributor holds a complete copy of the ACRES catalogue (image and data), a range of sample products, price lists, order forms and information material.

For details of the nearest distributor in your state, contact:

Australian Centre for Remote Sensing  
Dunlop Court, Fern Hill Park  
BRUCE ACT 2617  
PO Box 2 BELCONNEN ACT 2616

Telephone: (02) 6201 4201

Facsimile: (02) 6201 4366

<http://www.ga.gov.au/acres/index.jsp>

## **Australian Geological Survey Organisation (AGSO)**

AGSO provided gridded aeromagnetic, aeroradiometric and gravity grid values images. Located and gridded data for various areas of Australia are available from the Australian Geological Survey Organisation (AGSO) on magnetic tape. Orders for these images should be addressed to:

Australian Geological Survey Organisation (AGSO)  
GPO Box 378  
CANBERRA ACT 2601

Telephone: (02) 6249 9111

Facsimile: (02) 6249 9999

<http://www.agso.gov.au>

## **Department of Mineral Resources TAS**

Department of Mineral Resources provides point data and vector data with AMG coordinates of the Fingal Tier region. Open-file and contract data is available from:

Department of Mineral Resources  
Gordons Hill Road  
PO Box 56  
ROSNY PARK TAS 7018

Telephone: (03) 6233 8333

Facsimile: (03) 6233 8338

<http://www.mrt.tas.gov.au>

### **Earth Observation Satellite Company**

Earth Observation Satellite Company (EOSAT) operates the U. S. Landsat system of remote sensing satellites, and distributes data through a network of international ground stations, representatives, and field offices. Since 1972, Landsat satellites have acquired over 2.5 million multispectral images of the earth's surface. For information on EOSAT products and services, contact:

Space Imaging  
12076 Grant St.  
Thornton, CO 80241

Telephone: 1-303-254-2000

Facsimile: 1-303-254-2215

<http://www.spaceimaging.com>

### **Eurimage**

Eurimage is a distributor of satellite imagery and related products throughout the European community. They are a service oriented company that concentrates on meeting the growing need for environmental information. Eurimage provides products in both digital and photographic formats, including raw data, standard processed products, enhanced products, geocoded products, and cartographic maps derived from satellite imagery. For more information, contact:

Eurimage  
Viale E. D'Onofrio, 212  
00155 Rome, Italy

Telephone: +39-06-40 69 45 55

Facsimile: +39-06-40 69 42 31/32

<http://www.eurimage.com>

### **GeoImage Pty Ltd**

GeoImage are official distributors of ACRES Thematic Mapper and MSS digital and image products and offer advice on the best type of products for individual applications. They can also supply copies of digital data and slide sets of the Queensland Department of Resource Industries open-file geophysical data. Image products and digital data on any type of magnetic media can be purchased directly from:

GeoImage Pty Ltd  
13/180 Moggill Rd,  
Taringa, Brisbane QLD

P.O Box 789, Indooroopilly QLD 4068

Telephone: (07) 3871 0088

Facsimile: (07) 3871 0042

<http://www.geoimage.com.au>

**Intera Information  
Technologies Corporation**

Intera is a leader in providing spatial information solutions to petroleum and other resource industries, and to governments. Intera is a specialist in the acquisition and use of radar data for many different applications. For more information on Intera products and services, contact:

Intera  
5th Floor  
3609 South Wadsworth Blvd.  
Denver, Colorado, USA 80235

Telephone: (303) 985-9900  
Facsimile: (303) 985-4111

<http://www.hartpub.com/bg/vd/7c.htm>

**Kevron Geophysics Pty  
Ltd**

Kevron Aerial Surveys provides high resolution airborne magnetic and aeroradiometric geophysical data. Data and client surveys are available from:

Kevron Geophysics Pty Ltd  
10 Compass Road  
Jandakot Airport  
PERTH WA 6164

Telephone: (08) 9417 3188  
Facsimile: (08) 9417 3558

<http://www.kevron.com.au>

**National Geophysical  
Data Center**

NGDC is the archival center for the airborne geophysical and remote sensing images collected in 1988 along the eastern side of the Osgood Mountains, Humboldt County, Nevada USA, an area that includes a string of disseminated gold deposits known as the Getchell Trend. The digital data are available from:

National Geophysical Data Center  
NOAA, NESDIS (E/GC1)  
325 Broadway  
Boulder, Colorado USA

Telephone: (303) 497 6826  
Facsimile: (303) 497 6513

<http://www.ngdc.noaa.gov>

**RADARSAT International,  
Inc.**

RADARSAT International (RSI) processes and distributes data from the Landsat and SPOT satellites collected in Canada, and also distributes SPOT data collected anywhere in the world for the Canadian market. In addition, RSI is the sole source in North America for data from the European ERS-1 satellite. RADARSAT will also market and distribute data from Canada's RADARSAT satellite due to be launched in 1994. For more information on RSI data products, contact:

13800 Commerce Parkway  
MacDonald Dettwiler Building  
Richmond, British Columbia  
V6X 2W2 Canada

Telephone: (604) 244-0400  
Facsimile: (604) 244-0404

<http://www.rsi.ca>

### **Satellite Remote Sensing Services**

The Satellite Remote Sensing Services provides facilities for the acquisition, processing and analysis of remotely sensed data from earth resources satellites, weather satellites, airborne scanners and other sensors. For further information contact:

Department of Land Administration  
65 Brockway Road  
Floreat WA 6014

PO Box 471  
Wembley WA 6014

Telephone: (08) 9340 9330  
Facsimile: (08) 9383 7142

<http://www.rss.dola.wa.gov.au/>

### **SPOT Imaging Services Pty Ltd**

SPOT Imaging Services provides SPOT panchromatic and multispectral imaging products. Each SPOT scene represents a ground area of 60 km x 60 km. The resolution is 10 m in the panchromatic mode and 20 m in the multispectral mode. Scenes are distributed either in the form of magnetic tapes or as photographic products by SPOT Imaging Services and their distributor network.

SPOT Imaging Services also provides image maps and Digital Terrain Models derived from SPOT stereo data.

#### **Singapore**

SPOT ASIA  
73, Amoy Street,  
Singapore 0106  
Republic of Singapore

Telephone: (65) 227 55 82  
Facsimile: (65) 227 62 31

#### **Australia**

Spot Imaging Services Pty Ltd  
P O Box 197  
ST LEONARDS NSW 2065

Telephone: (02) 9906 1733  
Facsimile: (02) 9906 5109

<http://www.spotimage.com.au>

#### **USA**

SPOT Image Corporation  
1897 Preston White Drive  
RESTON, VA., 22091  
USA

Telephone: (703) 620 2200  
Facsimile: (703) 648 1813

Europe



16 bis, avenue Edouard-Belin  
31030 Toulouse cedex/France

Telephone: (33) 61 53 99 76  
Facsimile: (33) 61 28 18 59

### **SSC Satellitbild**

Swedish Space Corporation (SSC) Satellitbild markets and distributes high quality satellite imagery products and related services and support. Their products include satellite image maps, sensor composite satellite image maps, precision corrected satellite imagery, and DTMs. They also offer project and consultancy services in areas such as natural resources mapping and forest management planning. For more information, contact:

SSC Satellitbild  
P.O. Box 816  
S-981 28  
KIRUNA, Sweden

Telephone: +46 (0)980 671 00  
Facsimile: +46 (0)980 160 44

<http://www.zacom.se>

### **USGS**

The EROS Data Center receives, processes, and distributes earth-image data acquired by satellite and aircraft and investigates new uses for such data.

U. S. Geological Survey  
EROS Data Center  
Sioux Falls, SD 57 198

Telephone: (05) 594 6151

<http://edcwww.cr.usgs.gov>

USA state index maps and information about available aeromagnetic and aeroradiometric maps and profiles can be obtained from the Branch of Geophysics.

Branch of Geophysics  
U. S. Geological Survey  
Mail Stop 964  
Box 25046, Federal Center  
Denver, CO 80225

Telephone: (303) 236 1343

<http://www.usgs.gov/network/>



# Index

## Symbols

- # symbol in configuration files 10
- \$\$ALG Parameter
  - in Dynamic Links menu file 187
- & in menu files 163
- .bar toolbar files 161
- .dcf digitizer configuration files 194
- .dig digitizer session files 195
- .dtp digitizer type file 191
- .dtp digitizer type files 192
- .dxf files, linking to 8
- .erm menu files 161
- .ers file (raster header file) 8
- .erv file (vector header file) 8
- .ker filter files 135
- .ldd map composition files 167
- |ask colourchooser 296

## Numerics

- 24 bit RGB output of hardcopy engine 209

## A

- accelerator keys in menus 163
- ACRES 377
- add transform (type) 270
- adding and displaying a Dynamic Link 62
- adding Dynamic Links 60
- adding the menu option for Dynamic Links 65, 186
- Advanced Very High Resolution Radiometer sensor characteristics 359
- aeromagnetic data suppliers 377
- aeroradiometric data suppliers 377
- aeroradiometric geophysical data suppliers 379
- AGSO 377
- airborne geophysical data suppliers 379
- airborne magnetic data suppliers 379
- Airborne scanner Geoscan characteristics 360
- algorithm .alg file format 105, 113
- algorithm commands in scripts 263
- Algorithm types in map composition .ldd files 171
- AlgorithmImagePS in map composition .ldd files 174
- AllLightsOff
  - in .alg algorithm files 119
- AllowedValues Block
  - in map composition .ldd files 171

- Altek digitizers 193
- angles in parameter specifications 12
- ARC/INFO
  - example Dynamic Link PostScript generation program 89
  - linking to 8
- arguments
  - initialization program 68
  - PostScript generation program 70
- arguments for Dynamic Links 68
- Array
  - in .ker filter files 128
- arrays in parameter specifications 11
- ASCII format
  - for digitizer coordinates 196
- aspect ratio in hardcopy 211
- Atmospheric
  - in raster dataset header files 35
- AttitudeKappa
  - in raster dataset header files 36
- AttitudeOmega
  - in raster dataset header files 36
- AttitudePhi
  - in raster dataset header files 36
- Australian Centre for Remote Sensing 377
- Australian data suppliers 377
- Australian Geological Survey Organisation 377
- Author
  - in .alg algorithm files 113
- AverageHeight
  - in raster dataset header files 34
- AVHRR sensor characteristics 359

## B

- BackgroundColor
  - in .alg algorithm files 114
- BackgroundColour Block
  - in hardcopy .hc files 158
- BackgroundColourSet
  - in .alg algorithm files 114
- band\_range specifying for command line importing 215
- BandId Block
  - in raster dataset header files 29
- BandID Blocks
  - in .frm formula files 132
- Band-Interleaved-by Line data format 44, 213
- BandNumber
  - in .frm formula files 132
- BATCH
  - in menu and toolbar files 162

- batch scripting
  - actions 257
  - algorithm commands 263
  - batch script library 282
  - cell type 287
  - color keywords 279
  - controls 290
  - dialog boxes 248
  - documentation 225
  - ER Mapper objects 253
  - example script 229
  - example wizard script 234
  - file name, extension and path keywords 279
  - filter commands 273
  - flow control 290
  - formula commands 272
  - getting started 226
  - image manipulation 254
  - input in scripts 247
  - language and commands, see scripting language
  - layer commands 267
  - page setup commands 275
  - page size chooser 288
  - page view mode commands 276
  - preferences commands 277
  - print commands 280
  - run script from command line 228
  - see also wizards 239
  - status dialog 281
  - surface commands 265
  - transforms commands 270
  - using preferences to remember settings 251
  - variables 284
  - view mode commands 276
  - warning command 336
  - warning dialog 281
  - window commands 262
  - wizards 250
  - wizards page layout 236
- BaudRate
  - in .dcf digitizer configuration files 194
- Begin-End blocks 13
- BIL (Band-Interleaved-by Line) data format 44, 213
- blocks
  - file syntax 13
- Bottom
  - in .alg algorithm files 120
- BottomBorder
  - in .alg algorithm files 118
- BottomRightCorner Block
  - in .alg algorithm files 116
- BoundingBox in map composition .ldd files 174
- box syntax in vector data files 55
- button action command in batch scripting 294
- ButtonDown
  - in .dtp digitizer type file 192
- buttons
  - Color 62
  - Dataset chooser 61
  - Dynamic Link chooser 61
  - Dynamic Link layers 60
  - Edit 62
  - Open Annotation Editor 62
- ButtonUp
  - in .dtp digitizer type file 192
- ByteOrder
  - in raster dataset header files 23
  - in vector dataset header files 50

**C**

C

- code 135
- function arguments 147
- function format 144
- user defined function name 144

C function source code 143

C programming language in Dynamic Links 18

Calcomp digitizers 193

CalibrationDate
 

- in raster dataset header files 31

CALLBACK
 

- in menu and toolbar files 162

callback options available 163

CameraManufacturer
 

- in raster dataset header files 30

CameraModel
 

- in raster dataset header files 30

CanEdit
 

- in .alg algorithm files 126

CANVASHEIGHT argument in PostScript generation programs 92

CANVASWIDTH argument in PostScript generation programs 92

cartographic maps suppliers 378

Cell type in batch scripting 287

cell\_range specifying for command line importing 215

ChooseGcpsFromDigitizer
 

- in raster dataset header files 34

ChooseGcpsFromImage
 

- in raster dataset header files 34

chooser button for Dynamic Links 61

- chooser program for Dynamic Links 66
- choosing for Dynamic Links 62
- choosing the data source for a Dynamic Link 63, 187
- chroma correction in hardcopy 210
- CLibrarySwitches
  - in configuration file config.erm 104
- color
  - and Dynamic Links 73
  - creating color lookup tables 151
  - hardcopy 211
  - in hardcopy or printing 211
- color chooser button 62
- color compression in hardcopy or printing 209
- color lookup table files 155
- Color types in map composition .ldd files 172
- ColorMode
  - in .alg algorithm files 121
- Columns
  - in .ker filter files 128
- command line
  - running batch scripts 228
- Comments
  - in .alg algorithm files 113
  - in .dig digitizer session files 195
  - in .dtp digitizer type file 191
  - in ER Mapper files 10
  - in hardcopy .hc files 159
- compiling libermapper programs on PC 140
- compression of images in hardcopy 210
- ConfigFile
  - in .dig digitizer session files 195
- container blocks in wizards 358
- container right|left|justify 241
- Continuous mode for digitizers 196
- ControlPoints
  - in raster dataset header files 34
- ConvertToGreyscale
  - in .alg algorithm files 114
- convolution 135
- convolutions, See filters 127
- coordinate conversion utility programs 221
- coordinates in pixels in Dynamic Link Post-Script 101
- CoordinateSpace Block
  - in .alg algorithm files 115
  - in vector dataset header files 51
- CoordinateType
  - in .alg algorithm files 116
  - in raster dataset header files 24
  - in vector dataset header blocks 51
- Correction Block
  - in raster dataset header files 35

**D**

- data file
  - raster datasets 21
- data formats 15
- data source 62
  - choosing for Dynamic Links 63
  - defining in algorithms 105
- data suppliers in Australia 377
- data suppliers in Europe 378
- database
  - adding links to the menu file 183
- databases
  - links to 16
- DataBits
  - in .dcf digitizer configuration files 195
- DataFile
  - in raster dataset header files 23
  - in vector dataset header files 50
- Dataset
  - in .alg algorithm files 122
- Dataset Chooser
  - in Dynamic Links menu file 187
- Dataset chooser button 61
- DatasetHeader Block
  - in raster header files 21
  - in vector header files 49
- datasets
  - importing 15
  - raster data files 44
  - vector data files 54
  - vector header files 47
- DataSetType
  - in raster dataset header files 22
  - in vector dataset header files 49
- DataType
  - in raster dataset header files 22
  - in vector dataset header files 50
- dates in parameter specifications 12
- Datum
  - in .alg algorithm files 115
  - in raster dataset header files 24
  - in vector dataset header blocks 51
- datum
  - specifying for command line importing 219
- DBMS
  - Dynamic Links to 20, 59
- debugging Dynamic Links 73
- DefaultHardcopy
  - in configuration file config.erm 104
- DefaultHistStyle
  - in configuration file config.erm 104
- DefaultHSILUT
  - in configuration file config.erm 103
- DefaultRGBLUT

- in configuration file config.erm 103
- DefaultRtShade
  - in configuration file config.erm 104
- deleting files and directories in scripts 278, 308
- Department of Mineral Resources TAS data suppliers 377
- Description
  - in .alg algorithm files 113, 122, 123
  - in .frm formula files 131
  - in .ker filter files 127
  - in .lut lookup table files 156
  - in Dynamic Links menu file 185
  - in map composition .ldd files 169
- device DPI in PostScript 100
- dialog boxes, creating with scripts 248
- digitizer
  - configuration file .dcf format 194
  - coordinates 196
  - modes 196
  - session file .dig 195
  - support 196
- digitizer type file .dtp 191
- DigStringLength
  - in .dtp digitizer type file 191
- dimensions in points in Dynamic Links 97
- direct editing of external data using Dynamic Links 59
- disk2tape utility program 222
- DisplayBBox
  - in .alg algorithm files 119
- dithering in hardcopy or printing 209
- DoOptimisation
  - in configuration file config.erm 104
- DoStaticShade
  - in .alg algorithm files 122
- DotsPerInchAcross
  - in hardcopy .hc files 157
- DotsPerInchDown
  - in hardcopy .hc files 157
- dpiX PostScript variable 95
- dpiY PostScript variable 95
- DrawMode
  - in .alg algorithm files 118
- DTMs data suppliers 381
- dump\_cct utility program 222
- dynamic compilation on PC 135
- Dynamic Link Overlay Entries in .alg algorithm files 125
- Dynamic Link to ARC/INFO 89
- Dynamic Links 16
  - adding 60
  - adding the menu option 65, 186
  - arguments 68

- choosing a data source 62
- choosing the data source 66, 187
- creating 64
- debugging 73
- displaying the data 84
- example 75, 80, 89
- examples 20, 59
- extracting the data 64
- georeferenced overlays 60
- initializing 63, 66
- initializing the link 62
- inside 62
- introduction 59
- layer buttons 60
- link initialization program 67, 68
  - error reporting 69
  - example 69
- menu options 65
- multiple 59
- output of data chooser program 66
- output of link initialization program 68
- page relative overlays 60
- PostScript 64, 95, 97
  - generation program 70
    - error reporting 71
    - output 71
  - printing 72
  - procedure for displaying 62
  - Read/Write 59
  - supplied with ER Mapper 19
  - to DBMS 20
  - to vector files 20
  - using existing 62
- dynamiclinks.erm Dynamic Link menu file 186
- dynamiclinks.erm file format 65

**E**

- Earth Observation Satellite Company data suppliers 378
- EarthCurvature
  - in raster dataset header files 35
- ecw\_compress image compression command 222
- ecw\_compress\_gui image compression command 223
- edit button 62
- EDIT flag
  - in Dynamic Links menu file 186
- EDIT parameter in Dynamic Link menu file 65
- Edit program
  - in Dynamic Links menu file 186
- editable links 59
- editing filenames in scripts 278, 311

- EnforceAspectRatio
  - in configuration file config.erm 104
- environment variables
  - getting a value in scripts 278, 326
  - setting in scripts 278, 355
- EOSAT data suppliers 378
- ER Mapper raster translator 15
- erm\_arc\_layer Dynamic Link chooser program 89
- erm\_tapeutil utility program 221
- ERMConfig Block
  - in configuration file config.erm 103
- ermhe hardcopy engine 198
- erminit\_en0, example link initialization program 80
- ermmps\_arc Dynamic Link PostScript generation program 91
- ermmps\_example, PostScript generation program 76
- ermmps\_table\_circle, example Dynamic Link PostScript generation program 84
- EROS data suppliers 381
- errors
  - in Dynamic Link initialization programs 69
  - in Dynamic Link PostScript generation programs 71
- ERS-1 data suppliers 379
- ervecmacro.erm file 101
- Eurimage data suppliers 378
- European data suppliers 378
- example
  - algorithm file 105
  - batch script 229
  - complex dataset header file 39
  - dataset header file 38
  - Dynamic Link chooser program 89
  - Dynamic Link initialization program 69
  - Dynamic Link to ARC/INFO 89
  - Dynamic Links 59
  - file syntax 13
  - formula .frm file 131
  - Georeferenced dataset header file 39
  - hardcopy .hc file 157
  - hardcopy filter program 200
  - importing raster files 216
  - importing vector files 219
  - Landsat dataset header file 38
  - lookup table file 155
  - map composition .ldd files 178
  - map item .ldd file 169
  - PostScript 96
  - PostScript in Dynamic Links 99
  - Table of data shown as circles Dynamic Link 78
  - user Dynamic Link 75
  - vector header files 53
  - wizard script 234, 240
- Example User Dynamic Link menu option 75
- exit from batch scripts 311
- exiting in scripts 290
- ExposureCenterX
  - in raster dataset header files 36
- ExposureCenterY
  - in raster dataset header files 36
- ExposureCenterZ
  - in raster dataset header files 36
- Extents Block
  - in vector dataset header blocks 52, 53
- external data formats, linking to 8
- External Link Chooser
  - in Dynamic Links menu file 188
- EyeXYZ
  - in .alg algorithm files 119

**F**

- Far
  - in .alg algorithm files 120
- FFTInfo Block
  - in raster dataset header files 37
- Fiducial point block
  - in raster dataset header files 31
- FiducialInfo Block
  - in raster dataset header files 30, 31
- file exists 279
- file extensions 7
- file format
  - map composition .ldd files 167
  - menu files 161
  - raster files 7
  - toolbar files 161
  - vector files 7
- File name, extension and path keywords 312
- file syntax example 13
- file\_skip specifying for command line importing 216, 218
- FileExtent
  - in .alg algorithm files 126
- FileFormat
  - in vector dataset header blocks 52
- filmwriter specification files 157
- filter program
  - example 200
- filter programs in printing or hardcopy 197, 199
- FilterProgram
  - in hardcopy .hc files 158
- FilterProgramName

- in .alg algorithm files 126
- filters
  - C 129
  - convolution 135
  - defining in algorithms 105
  - file format 142
  - script commands 273
  - threshold filter 135
- Fixed Parameter
  - in Dynamic Links menu file 187
- fixed text syntax in vector data files 56
- FlagPosition
  - in .dtp digitizer type file 192
- FlyFar
  - in .alg algorithm files 120
- FlyFOV
  - in .alg algorithm files 120
- FlyNear
  - in .alg algorithm files 120
- Font type in map composition .ldd files 172
- Formula
  - in .alg algorithm files 123
  - in .frm formula files 131
- formula
  - functions 143
  - script commands 272
- Formula Sub-Blocks
  - in .alg algorithm files 123
- FormulaArg Blocks
  - in .alg algorithm files 124
  - in .frm formula files 131
- formulae
  - defining in algorithms 105
- free space, finding out in scripts 278, 319
- from72pt PostScript variable 97
- fromdevpt PostScript variable 97
- fromgeo utility program 221

**G**

- Gamma Block
  - in hardcopy .hc files 158
- gdt\_conv utility program 221
- Generate block
  - in map composition .ldd files 174
- geocoded data suppliers 378
- GeoImage Pty Ltd data suppliers 378
- geolinking
  - in scripts 263, 354
- geophysical data suppliers 378
- georeferenced data 60
- georeferenced layers 60
- Geoscan MSS Mark 2 sensor characteristics 360

- GIS
  - adding links to the menu file 183
  - Dynamic Links to 59
- GivenOrthoInfo Block
  - in raster dataset header files 36
- GMT (Greenwich Mean Time) in date specifications 12
- goto commands in scripts 290
- graphics format printing specification files 157
- gravity data suppliers 377
- Grid in map composition .ldd files 176
- GTCO digitizers 193

## H

- hardcopy
  - aspect ratio 211
  - chroma correction 210
  - color 211
  - color compression 209
  - configuring the /etc/printcap file 211
  - creating a filter program 199
  - device independent output 197
  - device specific format 210
  - dithering 209
  - ermhe engine 198
  - image compression 210
  - output format 199
  - output program 210
  - output size 211
  - processing and filter programs 197
  - processing stages 197
  - strip printing 198
  - subsampling and supersampling 211
- HardcopyInfo
  - in hardcopy .hc files 159
- header file
  - for raster and vector data 7
  - for vector datasets 47, 48
- HeaderOffset
  - in raster dataset header files 24
- HeadMatrix
  - in .alg algorithm files 119
- HeightI PostScript variable 95
- HRV sensor characteristics 372

## I

- if ... then ... else commands in scripts 290
- image compression in hardcopy 210
- image file compression 222
- images
  - importing 15
- importas2482 example command line importing 219



- importascii example command line importing 216
- importbil example command line importing 217
- importcct example command line importing 217
- importdisimp example command line importing 217
- importdola example command line importing 217
- importgeoimage example command line importing 219
- importing
  - command line switches 214, 218
  - raster and vector files 15, 213
  - raster file examples 216
  - vector file examples 219
- importusgs example command line importing 218
- including files in scripts 291
- InitialisationProgram
  - in .alg algorithm files 126
- initialization
  - arguments to program 68
  - example program 80
- Initialization program
  - in Dynamic Links menu file 186
- Initialization program for Dynamic Links 65
- initializing Dynamic Links 62, 63, 66, 67
- INPUTVECTOR 148
- integers in parameter specifications 11
- Intera data suppliers 379
- InterfaceType
  - in .ker filter files 129
- invert inverting utility program 213, 219
- IRS-IC sensor characteristics 364

## K

- Kernel Blocks
  - in .alg algorithm files 125
- Kernel Sub-Blocks
  - in .alg algorithm files 123
- Kevron Geophysics Pty Ltd data suppliers 379
- keywords
  - in parameter specifications 12

## L

- labels in scripts 290
- LANDSAT 4 sensor characteristics 367
- LANDSAT 5 sensor characteristics 369
- LANDSAT 7 sensor characteristics 372
- Landsat data suppliers 378, 379
- Landsat MSS data suppliers 378

- LANDSAT MSS sensor characteristics 367
- Landsat TM and Landsat MSS data suppliers 377
- LANDSAT TM sensor characteristics 369
- LastCharacter
  - in .dtp digitizer type file 191
- LastUpdated
  - in .alg algorithm files 114
  - in raster dataset header files 23
  - in vector dataset header files 50
- layers script commands 267
- Left
  - in .alg algorithm files 120
- LeftBorder
  - in .alg algorithm files 118
- LegendRule block
  - in map composition .ldd files 169
- LensSerialNr
  - in raster dataset header files 31
- libermapper programs, compiling on a PC 140
- Line mode for digitizers 196
- line\_range specifying for command line importing 215
- Linepath in map composition .ldd files 174
- LinePrinter
  - in configuration file config.erm 103
- Link chooser
  - in Dynamic Links menu file 186, 187
- link initialization program 68
- links
  - choosing a data source 63, 187
  - to DBMS 20
  - to vector files 20
- LinkType
  - in .alg algorithm files 126
- list all image files 297
- List types in map composition .ldd files 172
- LoadMethod
  - in .lut lookup table files 156
- lookup tables
  - creating 151
  - file format 155
- Lookup value for pseudocolor display 155
- LookupTable
  - in .alg algorithm files 121
- loops in scripts 291
- LUT
  - file entries 155
  - in .lut lookup table files 156

## M

- machine type, determining in scripts 287
- makelut, color lookup table generation 151

MANAGERELATIVEPATHS dynamic link chooser option 188  
 map composition .ldd files 167  
 map item object definition files 167  
 map\_box and map\_polygon object item definition files 167  
 map\_box syntax in vector data files 57  
 map\_polygon syntax in vector data files 57  
 MaterialAmbient  
   in .alg algorithm files 120  
 MaterialDiffuse  
   in .alg algorithm files 120  
 MaterialShininess  
   in .alg algorithm files 120  
 MaterialSpecular  
   in .alg algorithm files 120  
 MaximumDotsAcross  
   in hardcopy .hc files 158  
 MaximumDotsDown  
   in hardcopy .hc files 158  
 MaximumInputValue  
   in .alg algorithm files 124  
 MaximumOutputValue  
   in .alg algorithm files 125  
 menu .erm files 161  
 menu entries in Dynamic Links menu file 183  
 Menu option  
   in Dynamic Links menu file 185  
 menu options  
   adding for Dynamic Links 65  
   specifying keyboard selection keys 163  
 Menu/button option  
   in menu and toolbar files 161  
 MenuButtonDescription  
   in .alg algorithm files 125  
 mergeers merging utility program 213  
 MinimumInputValue  
   in .alg algorithm files 124  
 MinimumOutputValue  
   in .alg algorithm files 125  
 ModelMatrix  
   in .alg algorithm files 119  
 Monocolor PostScript 65, 73  
 Monocolour PostScript  
   effect on performance 101  
 MosaicType  
   in .alg algorithm files 114  
 MSS sensor characteristics 367  
 multispectral data suppliers 378, 380

## N

Name  
   in .alg algorithm files 113, 120, 123

  in .dcf digitizer configuration files 194  
   in .dig digitizer session files 195  
   in .frm formula files 131  
   in .ker filter files 127  
   in .lut lookup table files 156  
   in hardcopy .hc files 157  
   in map composition .ldd files 170  
   in raster dataset header files 23  
 National Geophysical Data Center data suppliers 379  
 navigation buttons container in wizards 238  
 Near  
   in .alg algorithm files 120  
 NGDC data suppliers 379  
 No Parameter  
   in Dynamic Links menu file 187  
 NOAA 359  
 NorthArrow in map composition .ldd files 176  
 NrEntries  
   in .lut lookup table files 156  
 Number types in map composition .ldd files 171

## O

object syntax in vector files 55  
 objects in vector files 55  
 OkOnSubsampledData  
   in .ker filter files 128  
 open standards for file and data formats 7, 18  
 operators in scripting language 291  
 Option separators  
   in menu and toolbar files 163  
 Oracle, linking to 8  
 orthorectification 27  
 output in scripts 280  
 output of Dynamic Links data chooser program 66  
 output of Dynamic Links initialization program 68  
 output of Dynamic Links PostScript generation program 71  
 output program for hardcopy 210  
 output size in hardcopy 211  
 OutputProgram  
   in hardcopy .hc files 158  
 oval syntax in vector data files 55

## P

page relative data 60  
 page relative layers 60  
 page setup  
   script commands 275  
 page size chooser in batch scripting 288

- page view mode 276
- page view mode, determining in scripts 286
- PageConstraint
  - in .alg algorithm files 117
- PageHeight
  - in .alg algorithm files 118
- PageSize Block
  - in .alg algorithm files 116
- PageViewMode
  - in .alg algorithm files 113
- PageWidth
  - in .alg algorithm files 117
- panchromatic data suppliers 380
- Paper Types
  - in hardcopy .hc files 159
- Parameter types in map composition .ldd files 171
- parameters
  - in Dynamic Links menu file 185
  - in ER Mapper files 10
  - syntax 10
- Parity
  - in .dcf digitizer configuration files 194
- performance and accuracy in PostScript Dynamic Links 100
- petroleum data suppliers 379
- photocopier specification files 157
- PlatformType
  - in raster dataset header files 30
- plotter specification files 157
- Point mode for digitizers 196
- point syntax in vector datasets 55
- polygon syntax in vector data files 56
- PolyLens
  - in raster dataset header files 35
- polyline syntax in vector data files 56
- Port
  - in .dcf digitizer configuration files 194
- PostScript
  - adding links to the menu file 183
  - commands
    - moveto 95
  - converting data in Dynamic Links 62
  - device DPI 100
  - dimensions in pixels 95
  - example code 96
  - file size effects on performance 100
  - functions in ER Mapper 98
  - in Dynamic Links 18, 64, 95
  - interpretation in Dynamic Links 95
  - linking to 8, 59
  - Monocolor or Truecolor 65, 73
    - effect on performance 101
  - performance and accuracy 100
    - using existing 101
  - variable canvas size 95
- variables 97
  - dpiX 95
  - dpiY 95
  - from72pt 97
  - fromdevpt 97
  - HeightI 95
  - WidthI 95
  - variables defined in ER Mapper 95
- PostScript conversion program 70
- PostScript generation program 64
- PostScript generation program call
  - in Dynamic Links menu file 185
- PostScript type
  - in Dynamic Links menu file 185
- PostScriptPrinter
  - in configuration file config.erm 103
- preferences
  - script commands 277
- printer specification files 157
- printing
  - output program 210
  - output size 211
  - processing and filter programs 197
  - processing stages 197
  - see also hardcopy 197
- printing Dynamic Links 72
- printing in scripts 280
- printing output format 199
- ProcessFirst in map composition .ldd files 174
- Program call
  - in menu and toolbar files 162
- Projection
  - in .alg algorithm files 116
  - in raster dataset header files 24
  - in vector dataset header blocks 51
- projection
  - specifying for command line importing 219
- pseudocolor
  - creating color lookup tables 151
  - lookup table files 155
- PSExecInclude Block
  - in map composition .ldd files 177
- PSLibInclude Block
  - in map composition .ldd files 177

## R

- radar data suppliers 379
- RADARSAT International, Inc. data suppliers 379
- RadialLens
  - in raster dataset header files 35

- raster datasets
  - data files 44
  - header file 21
  - header files 21
  - importing 15
- raster file format 7
  - integrating data 15
- raster image
  - importing 213
- raster images
  - header files 21
  - import switches 214
  - importing 15
  - importing examples 216
- Read/Write Dynamic Links 59
- real numbers in parameter specifications 11
- RegionInfo Block
  - in raster dataset header files 31
- RegionName
  - in raster dataset header files 32
- RegistrationCellX
  - in raster dataset header files 27
- RegistrationCellY
  - in raster dataset header files 27
- RegistrationCoordinate block
  - in raster dataset header files 28
- remote sensing data suppliers 379
- RGBcolour Block
  - in .alg algorithm files 126
- Right
  - in .alg algorithm files 120
- RightBorder
  - in .alg algorithm files 118
- rmoveto
  - PostScript command 95
- Rotation
  - in .alg algorithm files 116
  - in raster dataset header files 25, 34
  - in vector dataset header blocks 52
- rotation
  - specifying for command line importing 219
- Rows
  - in .ker filter files 128
- RSAC data suppliers 380
- run length limited image compression in hard-copy 210

**S**

- Satellite Remote Sensing Services data suppliers 380
- SCALE
  - in raster dataset header files 36
- Scale
  - in .alg algorithm files 118
- ScaleBar in map composition .ldd files 175
- Scalefactor
  - in .ker filter files 128
- ScreenAxesRotn
  - in .alg algorithm files 119
- scripting language
  - absolute|relative path 291
  - add (new) layer 267, 292
  - add (new) transform (to layer) 292
  - add filter 273
  - add filter (to layer) 292
  - add formula 272
  - add formula (to layer) 292
  - add height layer 244
  - add layer 267
  - add layer (to surface) 293
  - add surface (to algorithm) 265, 293
  - add transform (to layer) 270, 293
  - add transform point 270, 293
  - add transform to layer input 270, 293
  - algorithm2 = algorithm1 263, 294
  - ask 248
  - ask action 235, 241, 250, 294
  - ask bandchooser 249, 295
  - ask bandmenu 242, 249, 296
  - ask begin 231
  - ask colorchooser 249, 296
  - ask datum 249, 296
  - ask directory 249, 296
  - ask end 231
  - ask file 231, 249, 297
  - ask gridlayermenu 250, 297
  - ask hardcopy 250, 298
  - ask link 250, 298
  - ask listmenu 250, 299
  - ask listmenu\_exclusive 241, 249, 300
  - ask lutmenu 250, 301
  - ask projection 301
  - ask text|number 250, 301
  - ask yes/no 238, 250, 301
  - build absolute filespec 277, 302
  - build file path 277, 302
  - build relative filespec 277, 302
  - color keywords 302
  - color\_blue (keyword) 279, 302
  - color\_green (keyword) 279, 302
  - color\_red (keyword) 279, 302
  - concatenation 283
  - container above|below|left|right 241, 251, 303
  - container begin|end 235, 241, 251, 303
  - container items 241, 251, 304
  - container labels 251, 304

- container right|left justify 251
- container right|left|justify 303
- container width|height 241, 251, 304
- convert file separator 304
- convert to english|native 278
- copy algorithm 263, 305
- copy algorithm from window 235, 262, 305
- copy algorithm to window 232, 245, 262, 305
- copy filter 273, 305
- copy formula 272, 305
- copy layer 267, 306
- copy surface 265, 306
- copy transform 270, 306
- copy window 255, 262, 306
- current algorithm 254, 263, 306
- current filter 254, 273, 307
- current formula 254, 272, 307
- current input 254
- current layer 254, 267, 307
- current surface 254, 266, 307
- current transform 254, 270, 307
- current window 254, 262, 308
- delete algorithm 263, 308
- delete directory|file 278, 308
- delete filter 273, 308
- delete formula 272, 308
- delete layer 267, 309
- delete status (dialog) 282, 309
- delete surface 266, 309
- delete transform 270, 309
- delete window 262, 309
- dialog box
  - title 231, 248
- dirname (keyword) 279, 312
- duplicate filter 273, 310
- duplicate formula 272, 310
- duplicate layer 267, 310
- duplicate surface 266, 310
- duplicate transform 270, 310
- edit file 311
- edit text file 278
- error codes 291
- execute system command 279
- exists 311
- exit 290, 311
- exit batch process 278
- file exists 279
- fileext (keyword) 279, 312
- filename (keyword) 279, 312
- filter2 = filter1 312
- filterl2 = filterl1 273
- first|last|nex|previous surface 314

- first|last|next|previous algorithm 263, 312
- first|last|next|previous filter 273, 313
- first|last|next|previous formula 272, 313
- first|last|next|previous layer 267, 313
- first|last|next|previous surface 266
- first|last|next|previous transform 270, 314
- first|last|next|previous window 262, 314
- fit page to hardcopy 275, 314
- format number of digits after decimal point 279
- format value precision 315
- get (algorithm) page view mode 276, 324
- get (algorithm) view mode 277, 326
- get (ER Mapper) version 326
- get (page) contents topleft|bottomright 275, 315
- get algorithm coordsys 264, 315
- get algorithm datum|projection 264, 315
- get algorithm description 263, 315
- get algorithm layer count 264, 316
- get algorithm lut 264
- get algorithm lut (color table) 316
- get algorithm mode 264, 316
- get algorithm mosaic type 264, 316
- get algorithm topleft|bottomright (extents) 264, 316
- get algorithm units|rotation 264, 317
- get english file separator 278, 317
- get environment variable 278
- get ER Mapper version number 279
- get file size 278, 317
- get filter description 273, 317
- get filter matrix 273
- get filter params 273, 317
- get filter postsampled (flag) 273, 318
- get filter rows|cols 273, 318
- get filter scale|threshold (value) 274, 318
- get filter type 274, 318
- get filter userfile (filename) 274, 318
- get filter userfunc 274, 318
- get free space (on disk) 278, 319
- get layer azimuth|elevation 268, 319
- get layer band count 268, 319
- get layer band description 268, 319
- get layer cell size 268, 319
- get layer cell type 268, 320
- get layer color 267, 320
- get layer coordinates 268, 320
- get layer dataset (filename) 235, 268, 320
- get layer description 268, 320
- get layer edit|init program (name) 268,

- 321
- get layer editable (flag) 268, 321
- get layer formula 268, 321
- get layer input count 268
- get layer input filter count 268, 321
- get layer input transform count 268
- get layer link extension 268, 321
- get layer output filter count 268
- get layer output transform count 268, 322
- get layer shading 268, 322
- get layer type 268, 322
- get layer visibility 322
- get layer|\$lay input count 322
- get native path|file separator 278, 323
- get page constraints 275, 323
- get page scale 275, 323
- get page size 275, 323
- get page top|bottom|left|right border 275, 324
- get page topleft|bottomright (extents) 275, 323
- get page width|height 275, 324
- get preference 277, 324
- get preference (color) 277, 325
- get surface description 266, 325
- get surface layer count 266, 325
- get transform (limits) 325
- get transform input|output min|max 271
- get transform type 271
- getenv (environment variable) 326
- go algorithm 264, 326
- go background window 262, 327
- go window 262, 327
- goto 290
- if ... then ... else 232, 290
- include "filename" 291
- keywords 288
- labels 290
- layer active 268, 327
- layer2 = layer1 267, 327
- list contents of directory 278
- lsdir 278, 328
- load algorithm 232, 264, 328
- load filter (filename) 274, 328
- load formula (filename) 272, 329
- load layer formula (filename) 269, 329
- looping 291
- mathematical functions
  - abs(x) 284
  - asin(x), acos(x), atan(x) 283
  - ceil(x) 284
  - floor(x) 284
  - log(x), exp(x) 283
  - min(x,y), max(x,y) 284

- pow(x,y) 283
- sin(x), cos(x), tan(x) 283
- sqrt(x) 284
- move layer up|down|top|bottom 269, 329
- move surface up|down|top|bottom 266, 330
- new algorithm 264, 330
- new filter 274, 330
- new formula 272, 330
- new layer 269, 331
- new surface 266, 331
- new transform 271, 331
- new window 231, 262, 331
- next layer 269
- next layer (type) 332
- open status (dialog) 282, 332
- open window 262, 332
- operators 283
  - != > >= 283
  - # 283
  - () 283
  - = \* / + - % 283
  - ~ 283
- previous zoom 263, 358
- print 280, 333
- println 281, 333
- save algorithm 264, 334
- save algorithm as dataset 264, 334
- save algorithm as virtual dataset 264, 335
- save filter (filename) 274, 335
- save formula (filename) 272, 335
- save layer formula (filename) 269, 335
- say 231, 241
- say status 282, 336
- say warning 244, 281, 282, 336
- select algorithm 264, 336
- select filter 274, 336
- select formula 272, 337
- select layer 269
- select surface 266, 337
- select transform 271, 337
- select window 231, 263, 337
- set (formula) input to band 337
- set (page) contents bottomright from topleft 275, 338
- set (page) contents extents to algorithm extents 276, 338
- set (page) contents topleft|bottomright 275, 338
- set algorithm background (color) 264, 339
- set algorithm coordsys 264, 340
- set algorithm coordsys changeable to (flag) 338
- set algorithm datum|projection 264, 340

- set algorithm description 264, 340
- set algorithm lut (color table) 264, 341
- set algorithm mode 265, 341
- set algorithm mosaic type 265, 341
- set algorithm supersample type 265, 341
- set algorithm topleft|bottomright (extents) 265, 342
- set algorithm units|rotation 265, 342
- set environment variable 278
- set filter description 274, 342
- set filter matrix (element) 274, 343
- set filter params 274, 343
- set filter postsampled (flag) 274, 343
- set filter rows|cols 274, 343
- set filter scale|threshold 274, 344
- set filter type 274, 344
- set filter userfile (filename) 274, 344
- set filter userfunc (funcname) 274, 344
- set formula 272, 345
- set input to band 272
- set layer azimuth|elevation 269, 345
- set layer color 269, 345
- set layer dataset (filename) 244, 269, 346
- set layer description 244, 269, 346
- set layer edit|init program (name) 269, 346
- set layer editable (flag) 269, 346
- set layer formula 269, 347
- set layer input 245
- set layer input to band 269, 347
- set layer link extension 269, 347
- set layer link type to monocolour|truecolour 269, 347
- set layer shading on|off 269, 348
- set layer type 269, 348
- set output to filename|window 281, 333
- set page autovary\_value 275, 339
- set page center horizontal|vertical 275, 348
- set page constraints 275, 348
- set page extents from contents 276, 348
- set page scale 276, 349
- set page size 276, 349
- set page top|bottom|left|right border 276, 350
- set page topleft|bottomright (extents) 276, 350
- set page view mode 276, 350
- set page width|height 276, 350
- set pointer mode 263, 351
- set preference 277, 351
- set preference (color) 277, 351
- set surface description 266, 352

- set surface lut (color table) 266, 352
- set surface mode 266
- set surface mode (color) 352
- set surface name 266, 352
- set surface zscale|zoffset|transparency 266, 353
- set transform clip 271, 353
- set transform input|output min|max (limits) 271, 353
- set transform limits (actual or percentage) 271, 353
- set transform output limits to input limits 271
- set transform to gaussian equalize 271, 354
- set transform to histogram equalize 271, 354
- set transform type 271, 354
- set view mode 245, 277, 354
- set window geolink mode 232, 263, 354
- setenv (environment variable) 355
- show image 239, 241, 251
- show image (in container) 355
- split string into array 279, 356
- surface active 266, 356
- system command 356
- system command status dialog 279
- transform2 = transform1 270, 357
- turn layer on|off 269, 357
- turn surface on|off 266, 357
- variables 284
  - arrays 287
  - color 286
  - coordsys 285
  - filtertype 286
  - layertype 285
  - machinetype 287
  - mode 285
  - mosaictype 285
  - number 284
  - page size options 288
  - pageviewmode 286
  - references 286
  - string 284
  - supersampletype 286
  - transformtype 286
  - viewmode 286
  - yesno 285
- window2 = window1 262, 357
- wizard begin|end 241
- wizard close 251, 357
- wizardpage begin|end 251, 358
- zoom in|out 263, 358
- zoom to 263, 358

- SenseDate
  - in raster dataset header files 23
  - in vector dataset header files 50
- sensor\_type specifying for command line importing 216
- SensorInfo Block
  - in raster dataset header files 29
- SensorName
  - in raster dataset header files 23
- SensorType
  - in raster dataset header files 30
- set 341, 342
- Sharp JX730 example hardcopy filter program 200
- SourceDataset
  - in raster dataset header files 23
- splitting strings in scripts 279, 356
- SPOT data suppliers 379, 380
- SPOT Imaging Services Pty Ltd data suppliers 380
- SPOT sensor characteristics 372
- spreadsheets
  - links to 16, 59
- SSC Satellitbild data suppliers 381
- Stats Block
  - in raster dataset header files 32
- status dialog 281
- StatusPosition
  - in .dtp digitizer type file 192
- StopBits
  - in .dcf digitizer configuration files 194
- Stream Blocks
  - in .alg algorithm files 121
- StreamInput
  - in .frm formula files 131
- StreamInput Sub-Blocks
  - in .alg algorithm files 123
- StreamInputID
  - in .alg algorithm files 123
- String types in map composition .ldd files 171
- strings, splitting in scripts 279, 356
- sub-blocks
  - file syntax 13
- submenus
  - in the Dynamic Links menu 185
- Submenus in menu files 162
- SubRegion
  - in raster dataset header files 32
- subsampling and supersampling in hardcopy production 211
- SunAngle
  - in .alg algorithm files 122
- SunAzimuth
  - in .alg algorithm files 122

- SuperSampling
  - in .alg algorithm files 114
- Surface Blocks
  - in .alg algorithm files 120
- surface script commands 265
- system command, executing in scripts 279

## T

- Table of data shown as circles, example Dynamic Link 78
- tabular data
  - dynamic links to 59
- tape utility programs 221
- tape\_struct utility program 222
- tape2disk utility program 222
- testing Dynamic Links and import and export utilities 18
- text strings in parameter specifications 11
- ThreeDInfo Block
  - in.alg algorithm files 118
- threshold filter 135
- scripting language
  - match transform 271, 329
- togeo utility program 221
- Tool tips text
  - in menu and toolbar files 162
- toolbar .bar files 161
- Top
  - in .alg algorithm files 120
- TopBorder
  - in .alg algorithm files 118
- TopLeftCorner Block
  - in .alg algorithm files 116
- Transform Point Coordinate Block
  - in .alg algorithm files 125
- Transform Sub-Blocks
  - in .alg algorithm files 124
- transforms
  - defining in algorithms 105
  - script commands 270
- Transparency
  - in .alg algorithm files 121
- Truecolor PostScript 65, 73
  - effect on performance 101
- TurnRate
  - in .alg algorithm files 119
- Type
  - in .alg algorithm files 122, 124
  - in .ker filter files 128
  - in map composition .ldd files 170
  - in raster dataset header files 31
  - in vector dataset header blocks 52
- TypeFile



in .dig digitizer session files 195

## U

### Units

- in .alg algorithm files 116
- in .frm formula files 132
- in configuration file config.erm 103
- in raster dataset header files 25, 29
- in vector dataset header blocks 52

### UseAverageHeight

- in raster dataset header files 34

### user code

- C library switches 147
- filter file format 142
- filter object file 143
- linking 147
- on PC 135

### UserCodeFileName

- in .ker filter files 129

### UserFunctionName

- in .ker filter files 129

### UserParameter Block

- in map composition .ldd files 170

### USERSTATS 148

### USGS data suppliers 381

## V

### Value

- in .frm formula files 132
- in raster dataset header files 29

variable aspect ratio in PostScript Dynamic

### Links 97

variable text syntax in vector data files 56

variables in scripting language 284

### vector datasets

- data file 47, 54
- header file examples 53
- header files 47, 48
- importing 16, 213

### vector file format 7

- data integration 16

### vector files

- Dynamic Links to 20

### vector header files

- examples 53

### vector images

- command line import switches 218
- importing 16, 213

### vector objects

- in datasets 55
- syntax 55

### VectorInfo Block

- in vector dataset header blocks 52

### Velocity

- in .alg algorithm files 119

verify existence of specified file 279, 311

### Version

- in .alg algorithm files 113
- in .lut lookup table files 155
- in configuration file config.erm 103
- in raster dataset header files 22
- in vector dataset header files 49

### view mode

- script commands 276

view mode, determining in scripts 286

### ViewMatrix

- in .alg algorithm files 119

### ViewMode

- in .alg algorithm files 113

## W

warning command in batch scripting 336

warning dialog 281

### WarpControl Block

- in .dig digitizer session files 196
- in raster dataset header files 32

### WarpType

- in raster dataset header files 34

### Width

- in .frm formula files 132
- in raster dataset header files 29

WidthI PostScript variable 95

### WindowAspectRatio

- in .alg algorithm files 119

### wizards

- ask and show commands 238
- container block 237, 358
- container navigation buttons 238
- example script 234, 240
- general guidelines for creating 239
- information container 236
- navigation container 236
- navigation container buttons 238
- page layout 236
- show image command 239
- user input container 236
- WizardPage Block 237

## X

### Xdimension

- in raster dataset header files 28

### XPosition

- in .dtp digitizer type file 192

## Y

### Ydimension

in raster dataset header files 28  
YES/NO types in map composition .ldd files  
171  
YPosition  
in .dtp digitizer type file 192

## Z

ZOffset  
in .alg algorithm files 121  
zoom 358  
zooming in scripts 358  
ZScale  
in .alg algorithm files 121  
ZScale in map composition .ldd files 176



