

Metview

User Guide

Documentation for Metview Version 3.12.3

Last updated 10th August 2011

Metview

USING METVIEW

Metview

MACRO LANGUAGE

Metview

ICON REFERENCE

I- USING METVIEW

Introduction	1
<i>About Metview</i>	1
<i>Data Formats Supported</i>	2
<i>Metview Architecture - Desktops and Icons</i>	2
<i>Starting Metview</i>	3
The Metview Desktop	6
<i>The Metview Environment</i>	6
<i>Desktop Functionality</i>	7
<i>Desktop bar</i>	7
<i>Desktop menu</i>	9
<i>Desktop icon drawers</i>	9
<i>Creating A New Desktop</i>	13
<i>Managing Desktops</i>	14
<i>Closing Desktops and Exiting Metview</i>	15
Metview Icons	17
<i>Introduction</i>	17
<i>Creating Icons Implicitly</i>	17
<i>Creating Icons Explicitly</i>	18
<i>Getting the icon</i>	19
<i>Editing the icon</i>	19
<i>Saving the icon</i>	20
<i>Creating Links</i>	20
<i>Icon Selection and Icon Menus</i>	21
<i>Single selection and the icon menu</i>	22
<i>Multiple selection and the current selection menu</i>	22
<i>Icon menus</i>	23
<i>Operations on Icons</i>	24

<i>Icon Feedback</i>	26
<i>Overview</i>	26
<i>Icon status</i>	27
<i>The output action</i>	27
<i>Handling Icons on the Desktop</i>	27
<i>Moving icons</i>	28
<i>Copying icons</i>	28
<i>Deleting icons</i>	28
<i>Finding icons</i>	28
<i>Re-arranging icons</i>	28
<i>Re-sizing icons</i>	29
<i>Icon Storage - The Desktop Drawers</i>	29
<i>Overview</i>	29
<i>Customising the icon drawers</i>	30
<i>Handling the icons in desktop drawers</i>	31
<i>Icons Revealed</i>	32
<i>Icon Editors</i>	34
<i>Overview</i>	34
<i>The Icon Editor Window</i>	36
<i>Icon identification</i>	36
<i>Command buttons</i>	36
<i>Input area</i>	36
<i>Templates drawer</i>	37
<i>Command drawers</i>	37
<i>Save/Exit area</i>	37
<i>Editor Window Input Elements</i>	37
<i>Dependency between input parameters</i>	38
<i>Check buttons</i>	38
<i>Text fields</i>	38
<i>Option buttons</i>	39
<i>Icon fields</i>	40
<i>Colour selection list</i>	40
<i>Sliders</i>	41
<i>Editor Window Help Tools</i>	41
<i>I/O help tool</i>	42
<i>Geography help tool</i>	42

<i>Station help tool</i>	44
<i>Embedded Icons</i>	44
<i>Editor window icon field</i>	44
<i>Icon input drawers</i>	45
<i>Embedding an icon from the icon input drawer</i>	46
<i>Embedding an icon from a user desktop</i>	47
<i>Template Icons</i>	47
<i>Overview of template icons</i>	47
<i>The template icon drawer</i>	48
<i>Using template icons</i>	49
<i>Merging with desktop icons</i>	50
<i>Editing Icons in ASCII Mode</i>	50
<i>Working with Text Icon Editors</i>	51
<i>Handling text</i>	52
<i>Using alternative editors</i>	52
<i>Drops into text icon editors</i>	53
<i>Using Customised NEdit as the Alternative Editor</i>	54
<i>Syntax Highlighting</i>	55
<i>Calltips</i>	55
<i>Listing of built-in functions</i>	56
<i>Listing of available library macros</i>	56
<i>Code templates</i>	59
<i>Importing Icons with NEdit</i>	60
<i>Commenting blocks of code</i>	60
<i>Updating NEdit's Metview-specific files</i>	60
<i>Visualisation - an Overview</i>	62
<i>Introduction</i>	62
<i>Approaches to Visualisation Work</i>	63
<i>Data Unit - Input to Visualisation</i>	63
<i>Visual Definitions - Plotting Control</i>	64
<i>The View Concept in Visualisation</i>	65
<i>Direct Visualisation</i>	68
<i>Layout Visualisation</i>	69

<i>The Display Layout</i>	71
<i>Overview of Layout Design</i>	71
<i>The Display Window Icon Editor</i>	72
<i>Preparing Display Layouts</i>	74
<i>Setting size and orientation</i>	74
<i>Selecting frames</i>	74
<i>Sub-dividing frames</i>	75
<i>Frame numbering</i>	76
<i>Joining frames</i>	77
<i>Adding and deleting frames</i>	78
<i>Moving and re-sizing frames</i>	78
<i>Using grid control in frame moving and re-sizing</i>	79
<i>Frame alignment</i>	80
<i>Connecting and disconnecting frames</i>	80
<i>Specifying views</i>	82
<i>Empy space</i>	82
<i>Example Layout Preparation</i>	82
<i>Views and Visualisation</i>	86
<i>Role of Views in Visualisation</i>	86
<i>Geography and computation parameters in views</i>	87
<i>Data overlay control in view icons</i>	88
<i>Plot positioning and dimensions</i>	89
<i>Plot borders</i>	90
<i>The Geography Tool</i>	91
<i>Working with the Geography Tool</i>	94
<i>Overview</i>	94
<i>Zoom</i>	94
<i>Choice of projection</i>	94
<i>Vertical longitude</i>	95
<i>Extraction of geographical elements</i>	95
<i>Customising the geography tool</i>	95
<i>Display Window Functionality</i>	97
<i>Introduction</i>	97
<i>Functionality Overview</i>	98

<i>Menu bar</i>	98
<i>Command buttons</i>	98
<i>Frame menu</i>	99
<i>Contents drawer</i>	100
<i>Controls drawer</i>	100
<i>Using the Display Window</i>	101
<i>Point coordinates</i>	101
<i>Magnification</i>	101
<i>Zoom</i>	101
<i>Changing the display geography</i>	102
<i>Saving the display geography</i>	104
<i>Obtaining values at a location</i>	105
<i>Visualising stacks of plots</i>	106
<i>Animating stacks of plots</i>	110
<i>Converting a visualisation to a macro program</i>	111
<i>Printing a visualisation</i>	112
<i>Filenaming Conventions for JPEG and PNG Plots</i>	114
<i>Visualisation Input by Icon Drag and Drops</i>	114
<i>Data Icon Drops</i>	116
<i>Data overlay rules and overlay controls</i>	116
<i>Data overlay in practice</i>	117
<i>Plotting Geopoints Over Fieldsets</i>	119
<i>Observation Grouping</i>	119
<i>Visual Definition Icon Drops</i>	119
<i>View Icon Drops</i>	120
<i>Using the Contents Window</i>	121
<i>Overview</i>	121
<i>The contents structure</i>	121
<i>Assignment of visual definitions</i>	123
<i>Modifying a visualisation contents</i>	123
<i>Icon Drops in Contents</i>	124
<i>Metview Utilities</i>	126
<i>The Message Window</i>	126
<i>The Icon Output Window</i>	126
<i>The Metview Help</i>	127

The Metview Tools 128

Mail..... 129

News..... 130

Monitor 131

Stations..... 133

II- MACRO LANGUAGE

Macro Language Fundamentals	1
<i>Basic Syntax.....</i>	<i>1</i>
<i>Identifiers.....</i>	<i>1</i>
<i>Reserved keywords.....</i>	<i>1</i>
<i>Comments.....</i>	<i>2</i>
<i>Operators.....</i>	<i>2</i>
<i>Variables</i>	<i>3</i>
<i>Creating Variables</i>	<i>3</i>
<i>Lifetime of variables</i>	<i>4</i>
<i>Scope of variables.....</i>	<i>4</i>
<i>Modifying the scope of variables</i>	<i>5</i>
<i>Null variables and the nil keyword</i>	<i>6</i>
<i>Program Control.....</i>	<i>6</i>
<i>Tests - the if (and if/else) statement.....</i>	<i>6</i>
<i>Tests - the when statement.....</i>	<i>7</i>
<i>Tests - the case statement.....</i>	<i>7</i>
<i>Loops - the for statement</i>	<i>7</i>
<i>Loops - the while statement</i>	<i>8</i>
<i>Loops - the repeat statement.....</i>	<i>8</i>
<i>Loops - the loop statement.....</i>	<i>8</i>
Macro data types.....	9
<i>Finding the Type of an Expression</i>	<i>9</i>
<i>Strings.....</i>	<i>9</i>
<i>Numbers.....</i>	<i>11</i>
<i>Dates.....</i>	<i>11</i>
<i>Creating</i>	<i>11</i>
<i>Dates in Metview MARS requests.....</i>	<i>12</i>
<i>Converting dates to strings and numbers</i>	<i>13</i>
<i>Loops with dates</i>	<i>16</i>
<i>Lists.....</i>	<i>16</i>
<i>Areas</i>	<i>17</i>

<i>Definitions</i>	17
<i>Limitations of Definitions</i>	19
<i>Fieldsets</i>	19
<i>How operators and functions work on fieldsets</i>	19
<i>Indexing fieldsets</i>	20
<i>Merging fieldsets</i>	21
<i>Notes on fieldset calculations</i>	22
<i>Observations</i>	22
<i>Geopoints</i>	23
<i>Format details</i>	23
<i>Storing Data Origin Information in a Geopoints File</i>	25
<i>Operations between geopoints and fieldsets or images</i>	25
<i>Missing values in geopoints</i>	25
<i>NetCDF</i>	26
<i>Deriving NetCDF output in Metview Macro</i>	26
<i>How operators and functions work on NetCDF</i>	27
<i>Working with multi-variable NetCDF files</i>	28
<i>Accessing individual NetCDF values</i>	29
<i>Satellite Images</i>	30
<i>Arrays - Vectors and Matrices</i>	30
<i>Input and Output in Macro</i>.....	32
<i>Input : read() Function</i>	32
<i>Overview</i>	32
<i>GRIB input</i>	33
<i>ASCII matrix input</i>	33
<i>BUFR input</i>	33
<i>Geopoints input</i>	34
<i>NetCDF input</i>	34
<i>General ASCII input</i>	34
<i>Data Filtering</i>	35
<i>Filtering fieldsets (GRIB data)</i>	36
<i>Filtering observations (BUFR data)</i>	37
<i>Filtering geopoints</i>	37
<i>Output : write() and append() Functions</i>	37
<i>Overview</i>	37

<i>File handlers or file names?</i>	38
<i>Writing and appending</i>	39
<i>GRIB output</i>	40
<i>BUFR output</i>	40
<i>Geopoints output</i>	41
<i>NetCDF output</i>	41
<i>ASCII output</i>	42
<i>ASCII console output</i>	43
<i>Functions in Macro</i>	44
<i>Macro Function Essentials</i>	44
<i>Declaring functions</i>	44
<i>Calling functions and passing arguments</i>	45
<i>Scope of functions</i>	45
<i>Function look-up</i>	45
<i>Recursion</i>	47
<i>Building a Library of Functions</i>	47
<i>Using macro templates</i>	47
<i>Using macro inclusion</i>	47
<i>Using function look-up</i>	48
<i>External Functions</i>	48
<i>Icon-Functions</i>	50
<i>Using icon-functions</i>	51
<i>Translating icons to macro icon-functions</i>	51
<i>Information on icon-function input</i>	52
<i>Special Functions</i>	53
<i>Functions on functions</i>	53
<i>Handlers</i>	53
<i>Visualisation in Macro</i>	55
<i>Overview</i>	55
<i>The plot() Function</i>	56
<i>Overview of Visualisation Layout in Macro</i>	58
<i>Visualisation layout components</i>	58
<i>Macro layout implementation</i>	59
<i>Display Window Specification</i>	60

<i>Layout Frame Specification</i>	61
<i>Plot Subframe Specification</i>	63
<i>View Specification</i>	64
<i>Fast Layout Specification in Macro</i>	66
<i>A macro function for layouts of MxN frames</i>	67
<i>A macro function for layouts of MxN subframes</i>	68
<i>A macro function for general regular layouts</i>	69
<i>Using the Display Window editor</i>	71
<i>Assigning Data to Frames and Subframes</i>	71
<i>Output Formats and Destinations</i>	73
<i>Overview</i>	73
<i>Specifying output and destination</i>	73
<i>Forcing a New Page</i>	76
<i>Using FORTRAN in Macro</i>.....	77
<i>Overview</i>	77
<i>General Approach</i>	77
<i>Inlined or External</i>	78
<i>Interface routines</i>	78
<i>GRIB_API routines</i>	78
<i>GRIBEX() routine</i>	79
<i>The Macro-FORTRAN Interface (MFI) Routines</i>	79
<i>Getting arguments</i>	80
<i>Setting results</i>	80
<i>Handling fieldsets</i>	80
<i>Utility routines</i>	81
<i>Example of FORTRAN Function</i>	81
<i>The Legacy FORTRAN Interface Routines</i>	83
<i>Getting arguments</i>	84
<i>Setting results</i>	84
<i>Handling fieldsets</i>	84
<i>Utility routines</i>	85
<i>Example of FORTRAN Function</i>	85
<i>Implementing FORTRAN Functions</i>	87
<i>Implementation Example</i>	88

<i>Debugging the FORTRAN Functions</i>	90
<i>Using C/C++ in Macro</i>	91
<i>Overview</i>	91
<i>Writing Macro functions in C/C++</i>	91
<i>Numbers</i>	91
<i>Text Strings</i>	92
<i>Vectors</i>	92
<i>Input Fieldsets</i>	92
<i>Output Fieldsets</i>	93
<i>Miscellaneous</i>	94
<i>Macro Examples Using the C Interface</i>	94
<i>Example 1 - C++</i>	94
<i>Example 2 - C</i>	95
<i>Example 3 - C++</i>	96
<i>Inlined or External</i>	98
<i>Interface to the Metview system</i>	99
<i>Macro Run-Modes</i>	99
<i>Overview</i>	99
<i>Macro returns to the Metview interface</i>	100
<i>Run-mode dependent outcomes</i>	100
<i>Using a Data Cache in Macro</i>	101
<i>Exiting a Macro Program</i>	103
<i>Interface to the UNIX System</i>	104
<i>Controlling Macro Program Runs</i>	104
<i>Access to the Shell</i>	104
<i>Access to Environment Variables</i>	104
<i>Building a Macro User Interface</i>	105
<i>List of Input Element Functions</i>	105
<i>Using the dialog() Function</i>	109
<i>Using Macro Parameters</i>	110

<i>dialog() or Macro Parameters?</i>	111
<i>Metview in Batch mode</i>	113
<i>Running Macro Programs in Batch</i>	113
<i>Passing Arguments in Batch</i>	113
<i>A Batch Example</i>	114
<i>Performance and Timing</i>	117
<i>Stopwatch Functions</i>	117
<i>Using the Stopwatch with MARS Retrievals</i>	117
<i>Example of Using the Stopwatch Functions</i>	117
<i>List of Operators and Functions</i>	119
<i>Information Functions</i>	121
<i>The nil Operand</i>	123
<i>Functions and Operators on Numbers</i>	125
<i>Functions and Operators on Strings</i>	129
<i>Functions and Operators on Dates</i>	133
<i>Functions and Operators on Lists</i>	137
<i>Functions and Operators on Fieldsets</i>	139
<i>Functions and Operations on Images</i>	163
<i>Functions and Operators on Observations</i>	167
<i>Functions and Operators on Geopoints</i>	169
<i>Functions and Operators on NetCDF</i>	179
<i>Functions and Operators on Vectors and Matrices</i>	183
<i>Functions on Definitions</i>	185
<i>File I/O Functions</i>	187
<i>Timing Functions</i>	189
.....	189
<i>Metview Icon-Functions</i>	191
<i>Auxiliary functions</i>	191
<i>Data access and data filtering icon-functions</i>	192

<i>Data processing icon-functions</i>	192
<i>Data plotting icon-functions</i>	194
<i>Visualisation icon-functions</i>	196
<i>Vis5D icon-functions</i>	198
<i>Visual definition icon-functions</i>	198
<i>UNIX Interfacing Functions</i>	201
<i>Macro System Functions</i>	203

III- ICON REFERENCE

Introduction	1
Folder	3
<i>Actions on the Folder icon</i>	3
Notes	5
<i>The Notes Editor</i>	5
Shell Script	7
<i>The Shell Editor</i>	7
<i>Actions on the Shell icon</i>	7
Web Access	9
<i>The Web Access Editor</i>	9
<i>Actions on the Web Access Icon</i>	9
Display Window	11
MARS Retrieval	13
<i>Brief Overview of MARS</i>	13
<i>Data Formats</i>	13
<i>Contents of MARS</i>	13
<i>Spatial Coordinate Systems</i>	16
<i>Postprocessing</i>	17
<i>Retrieval Efficiency</i>	17
<i>The MARS Retrieval Editor</i>	18
<i>Help in Specifying Retrievals</i>	28
<i>WebMARS</i>	28
<i>Actions on the MARS Retrieval Icon</i>	29
<i>Saving MARS Retrieval Output (as Files)</i>	30

<i>Examining GRIB Data</i>	31
<i>LatLong Matrix</i>	33
<i>From ASCII Data File to LLMatrix Icon</i>	33
<i>Adding a Header to an ASCII Data File</i>	33
<i>LLMatrix Header Components</i>	34
<i>Actions on the LLMatrix Icon</i>	36
<i>Matrix</i>	37
<i>The Matrix Editor</i>	37
<i>Actions on the Matrix icon</i>	38
<i>ECFS (ECMWF only)</i>	39
<i>The ECFS Editor</i>	39
<i>Actions on the ECFS icon</i>	39
<i>ODB Database (ECMWF only)</i>	41
<i>The ODB Database Editor</i>	41
<i>Actions on the ODB Database icon</i>	42
<i>The ODB Browser</i>	42
<i>ODB Access (ECMWF only)</i>	43
<i>The ODB Access Editor</i>	43
<i>Actions on the ODB Access icon</i>	49
<i>GRIB Filter</i>	51
<i>The GRIB Filter Editor</i>	51
<i>Actions on the GRIB Filter Icon</i>	52
<i>Observation Filter</i>	53
<i>Filtering Efficiency</i>	53
<i>The Observation Filter Editor</i>	54

<i>Actions on the Observation Filter Icon</i>	<i>56</i>
<i>Geopoints To GRIB</i>	<i>59</i>
<i>The Geopoints To GRIB Editor.....</i>	<i>59</i>
<i>Actions on the Geopoints To GRIB icon.....</i>	<i>60</i>
<i>GRIB To Geopoints</i>	<i>61</i>
<i>The GRIB To Geopoints Editor.....</i>	<i>61</i>
<i>Actions on the GRIB To Geopoints icon.....</i>	<i>61</i>
<i>GeoTools</i>	<i>63</i>
<i>The GeoTools Editor.....</i>	<i>64</i>
<i>Actions on the GeoTools icon</i>	<i>65</i>
<i>Station</i>	<i>67</i>
<i>The Station Editor.....</i>	<i>67</i>
<i>Actions on the Station Icon</i>	<i>68</i>
<i>Simple Formula.....</i>	<i>71</i>
<i>The Simple Formula Editor</i>	<i>72</i>
<i>Actions on the Simple Formula Icon</i>	<i>74</i>
<i>Formula</i>	<i>75</i>
<i>The Formula Editor.....</i>	<i>75</i>
<i>Actions on the Formula Icon</i>	<i>75</i>
<i>Macro</i>	<i>77</i>
<i>Macro Parameters</i>	<i>79</i>
<i>The Macro with Parameters Editor.....</i>	<i>79</i>
<i>NEDIT Syntax Highlight.....</i>	<i>81</i>
<i>NEDIT Syntax Highlight Details</i>	<i>81</i>

<i>Actions on the NEDIT Syntax Highlight Icon.....</i>	82
<i>NEDIT Syntax Highlight 2.0.....</i>	83
<i>NEDIT Syntax Highlight 2.0 Details.....</i>	83
<i>NEdit's configuration files.....</i>	84
<i>Changes to NEdit's configuration files.....</i>	84
<i>Additional customisations for NEdit.....</i>	84
<i>Actions on the NEDIT Syntax Highlight 2.0 Icon.....</i>	84
<i>Average Data.....</i>	87
<i>The Average Data Editor.....</i>	87
<i>Actions on the Average Data Icon.....</i>	88
<i>Cross-Section Data.....</i>	89
<i>The Cross-Section Data Editor.....</i>	89
<i>Actions on the Cross-Section Data Icon.....</i>	91
<i>Hovmøller Data.....</i>	93
<i>The Hovmøller Data Editor.....</i>	93
<i>Actions on the Hovmøller Data Icon.....</i>	96
<i>Percentile.....</i>	97
<i>The Percentile Data Editor.....</i>	97
<i>Actions on the PercentileIcon.....</i>	98
<i>Vertical Profile Data.....</i>	99
<i>The Vertical Profile Data Editor.....</i>	99
<i>Actions on the Vertical Profile Data Icon.....</i>	100
<i>Vectors.....</i>	101
<i>The Vectors Editor.....</i>	101
<i>Actions on the Vectors Icon.....</i>	102

Relative Humidity	103
<i>The Relative Humidity Editor</i>	103
<i>Actions on the Relative Humidity Icon</i>	103
Potential Temperature.....	105
<i>The Potential Temperature Editor</i>	105
<i>Actions on the Potential Temperature Icon</i>	106
VelocityPotential/StreamFunction	107
<i>The Velocity Potential / Stream Function Editor</i>	107
<i>Actions on Velocity Potential/Stream Function Icon</i>	108
RotationalWind/DivergentWind	109
<i>The Rotational Wind / Divergent Wind Editor.....</i>	109
<i>Actions on RotationalWind/DivergentWind Icon.....</i>	110
Custom Metgram	111
<i>The Metgram Editor</i>	111
<i>Actions on the Metgram Icon.....</i>	111
Time Series	113
<i>The Time Series Editor</i>	113
<i>Actions on the Time Series Icon.....</i>	114
Meteogram (ECMWF only).....	115
<i>The Meteogram Editor.....</i>	115
<i>Actions on the Meteogram Icon.....</i>	116
<i>Macro Example of Meteogram Icon.....</i>	116
Metgram Manager (Deprecated)	117
Tephigram Data.....	119

<i>The Tephigram Data Editor</i>	119
<i>Actions on the Tephigram Data icon</i>	121
Curve	123
<i>The Curve Editor</i>	123
<i>Actions on the Curve icon</i>	124
Budget	125
<i>The Budget Editor</i>	125
<i>Actions on the Budget Icon</i>	126
Spectra	127
<i>The Spectra Editor</i>	127
<i>Actions on the Spectra Icon</i>	128
Scores	129
<i>The Scores Editor</i>	129
<i>Actions on the Scores Icon</i>	129
TrajComp	131
<i>TrajComp Default Input Data</i>	131
<i>TrajComp Custom Input Data</i>	131
<i>The TrajComp Editor</i>	132
<i>Actions on the TrajComp Icon</i>	133
TrajPlot-x.x	135
<i>The TrajPlot-x.x Editor</i>	135
<i>Actions on the TrajPlot-x.x Icon</i>	136
Data Coverage (ECMWF only)	137
<i>Brief Overview</i>	137
<i>The Data Coverage Editor</i>	137

<i>Actions on the Data Coverage Icon.....</i>	<i>138</i>
<i>Average View</i>	<i>139</i>
<i>The Average View Editor</i>	<i>139</i>
<i>Actions on the Average View Icon</i>	<i>140</i>
<i>Cross Section View</i>	<i>141</i>
<i>The Cross-Section View Editor</i>	<i>141</i>
<i>Actions on the Cross SectionView Icon.....</i>	<i>143</i>
<i>Hovmøller View</i>	<i>145</i>
<i>The Hovmøller View Editor</i>	<i>145</i>
<i>Actions on the Hovmøller View Icon</i>	<i>146</i>
<i>Curve View.....</i>	<i>147</i>
<i>The Curve View Editor</i>	<i>147</i>
<i>Actions on the Curve View Icon.....</i>	<i>148</i>
<i>Empty View</i>	<i>149</i>
<i>The Empty View Editor</i>	<i>149</i>
<i>Actions on the Empty View Icon</i>	<i>150</i>
<i>Import View</i>	<i>151</i>
<i>The Import View Editor</i>	<i>151</i>
<i>Actions on the Import View Icon.....</i>	<i>152</i>
<i>Map View</i>	<i>153</i>
<i>The Map View Editor</i>	<i>153</i>
<i>Actions on the Map View Icon</i>	<i>155</i>
<i>Satellite View</i>	<i>157</i>
<i>The Satellite View Editor</i>	<i>157</i>
<i>Actions on the Satellite View Icon</i>	<i>159</i>

<i>Tephigram View</i>	161
<i>The Tephigram View Editor</i>	161
<i>Actions on the Tephigram View Icon</i>	163
<i>Text View</i>	165
<i>The Text View Editor</i>	165
<i>Actions on the Text View Icon</i>	166
<i>Vertical Profile View</i>	167
<i>The Vertical Profile View Editor</i>	167
<i>Actions on the Vertical Profile View Icon</i>	169
<i>Grib to Vis5D</i>	171
<i>The Grib to Vis5D Editor</i>	171
<i>Actions on the Grib To Vis5D Icon</i>	171
<i>Vis5D Object</i>	173
<i>The Vis5D Object Editor</i>	173
<i>Actions on the Vis5D Object Icon</i>	174
<i>Vis5D Script</i>	175
<i>Vis5D Trajectory</i>	177
<i>The Vis5D Trajectory Editor</i>	177
<i>Actions on the Vis5D Trajectory Icon</i>	178
<i>Vis5D Wind Object</i>	179
<i>The Vis5D Wind Object Editor</i>	179
<i>Actions on the Vis5D Wind Object Icon</i>	180
<i>Vis5D Window</i>	181
<i>The Vis5D Window Editor</i>	181

<i>Actions on the Vis5D Window Icon.....</i>	<i>181</i>
<i>Contour.....</i>	<i>183</i>
<i>Main Contour Features</i>	<i>183</i>
<i>The Contour Editor.....</i>	<i>183</i>
<i>Actions on the Contour Icon.....</i>	<i>194</i>
<i>Wind Plot</i>	<i>195</i>
<i>The Wind Plot Editor</i>	<i>195</i>
<i>Actions on the Wind Plot Icon</i>	<i>199</i>
<i>Observation Plot</i>	<i>201</i>
<i>Main Observation Plot Features</i>	<i>201</i>
<i>The Observation Plot Editor.....</i>	<i>201</i>
<i>General specification of Observation plotting.....</i>	<i>202</i>
<i>Plotting elements for Observation</i>	<i>203</i>
<i>Plotting elements for Feedback (Fb)</i>	<i>203</i>
<i>Actions on the Observation Plot Icon.....</i>	<i>206</i>
<i>Coastlines.....</i>	<i>207</i>
<i>The Coastlines Editor</i>	<i>207</i>
<i>Actions on the Coastlines Icon</i>	<i>208</i>
<i>Text Plot.....</i>	<i>211</i>
<i>The Text Plot Editor.....</i>	<i>211</i>
<i>Actions on the Text Plot Icon.....</i>	<i>214</i>
<i>Annotation</i>	<i>215</i>
<i>The Annotation Editor</i>	<i>215</i>
<i>Actions on the Annotation Icon</i>	<i>216</i>
<i>Symbol.....</i>	<i>217</i>
<i>The Symbol Editor</i>	<i>217</i>

<i>Actions on the Symbol Icon</i>	<i>219</i>
<i>Graph</i>	<i>221</i>
<i>The Graph Editor.....</i>	<i>221</i>
<i>Actions on the Graph Icon.....</i>	<i>223</i>
<i>Axis</i>	<i>225</i>
<i>The Axis Editor</i>	<i>225</i>
<i>Actions on the Axis Icon</i>	<i>228</i>
<i>Image Table</i>	<i>229</i>
<i>The Image Table Editor</i>	<i>229</i>
<i>Actions on the Image Table Icon</i>	<i>230</i>
<i>Legend Entry</i>	<i>231</i>
<i>The Legend Entry Editor</i>	<i>231</i>
<i>Actions on the Legend Entry Icon.....</i>	<i>232</i>
<i>Import Plot.....</i>	<i>233</i>
<i>The Import Plot Editor</i>	<i>233</i>
<i>Actions on the Import Plot Icon.....</i>	<i>233</i>
<i>Drawing Priority.....</i>	<i>235</i>
<i>The Drawing Priority Editor</i>	<i>235</i>
<i>Actions on the Drawing Priority Icon</i>	<i>235</i>
<i>Overlay Control</i>	<i>237</i>
<i>The Overlay Control Editor.....</i>	<i>237</i>
<i>Actions on the Overlay Control Icon.....</i>	<i>237</i>

I- USING METVIEW

INTRODUCTION

About Metview

Metview is an interactive meteorological application, which enables operational and research meteorologists to :

- access
- manipulate
- visualise

meteorological data on UNIX workstations. Metview has been designed as a flexible, modular and extendable system which will be able to accommodate the evolving needs of the user.

Metview is a cooperative project between ECMWF and INPE/CPTEC, Brazil. ECMWF has also been assisted by a staff member of Météo-France. The development of Metview started in 1991 and the first release became available to internal ECMWF users in December 1993. Metview was released to ECMWF Member States in October 1995. In mid 1999 an extensively revised Metview (version 2.0) was released comprising major improvements and increased flexibility. Version 3.0 introduced a new user interface and a new visualisation module to Metview in late 2000. In 2008, version 3.11 introduced a completely revised GRIB handling system, using the GRIB_API library to support both GRIB editions 1 and 2.

At ECMWF Metview is used on Linux (SuSE) and IBM (AIX) systems. It has also been installed and performs operational work on SGI, SUN, HP and DEC workstations in several Meteorological Offices. It has also been installed on other flavours of Linux, including Fedora and Ubuntu.

Metview is built and wrapped around two powerful and reliable packages - the ECMWF standard for data access (MARS - Meteorological Archiving and Retrieval System) and the ECMWF graphics package (MAGICS - Meteorological Applications Graphics Integrated Colour System). The user interface is based on MOTIF and the X Window System. Metview is a fully distributed system where modules can run on different UNIX workstations and servers. The coastline data used by Metview is the GSHHS - see Wessel, P., and W. H. F. Smith, A Global Self-consistent, Hierarchical, High-resolution Shoreline Database, J. Geophys. Res., 101, #B4, pp. 8741-8743, 1996.

This manual covers the newest version of Metview, Metview 3. It offers a completely re-designed interface and a completely new visualisation module with much enhanced layout capabilities.

Data Formats Supported

Meteorological data can be retrieved by Metview from the ECMWF MARS archive, from a local Meteorological Service archive or from a BUFR/GRIB file on a user disk. The supported data formats are :

- WMO GRIB (editions 1 and 2) format for fields
- WMO BUFR format for observations
- The extended WMO GRIB format for satellite images
- ASCII matrices of lat-long gridded data
- ASCII lists of irregularly spaced data points (Geopoints)
- NetCDF for data other than model fields and point data (e.g. cross sections)

Metview Architecture - Desktops and Icons

Metview aims to create a complete interactive working environment for the research and operational meteorologist and its functionality addresses its two core tasks :

- **Computations with meteorological data** - data in one of the allowed formats can be handled and processed using icon-based applications or Metview's own powerful macro language and the output saved in one of the allowed formats
- **Visualisation of meteorological parameters** - data in one of the allowed formats can be visualised as contoured geographical fields, cross-sections, zonal averages, vertical profiles, time series, tephigrams, Hovmöller diagrams, etc, on screen, on file, on paper. This data can be retrieved from a database or be the result of some previous computation

So Metview can be described both as a *meteorological desktop plotting package* and a powerful *meteorological data-processing software*.

The Metview functionality is based on two core concepts :

- the **Icon**
- and
- the **Desktop**

Metview represents all your work tools and components - scraps of text, UNIX shell scripts, model fields, database requests, observations, data filters, plotting specifications, contouring definitions, directories, symbolic links, etc - by means of interactive icons. This is the first Metview principle :

Everything in Metview is an icon

Since everything in Metview is an icon, it follows that any Metview task - be it of computation or visualisation or plain file management - is carried out as a sequence of operations on icons. Hence the second Metview principle :

Every Metview task is a sequence of operations on icons

A METVIEW DESKTOP POPULATED WITH ICONS

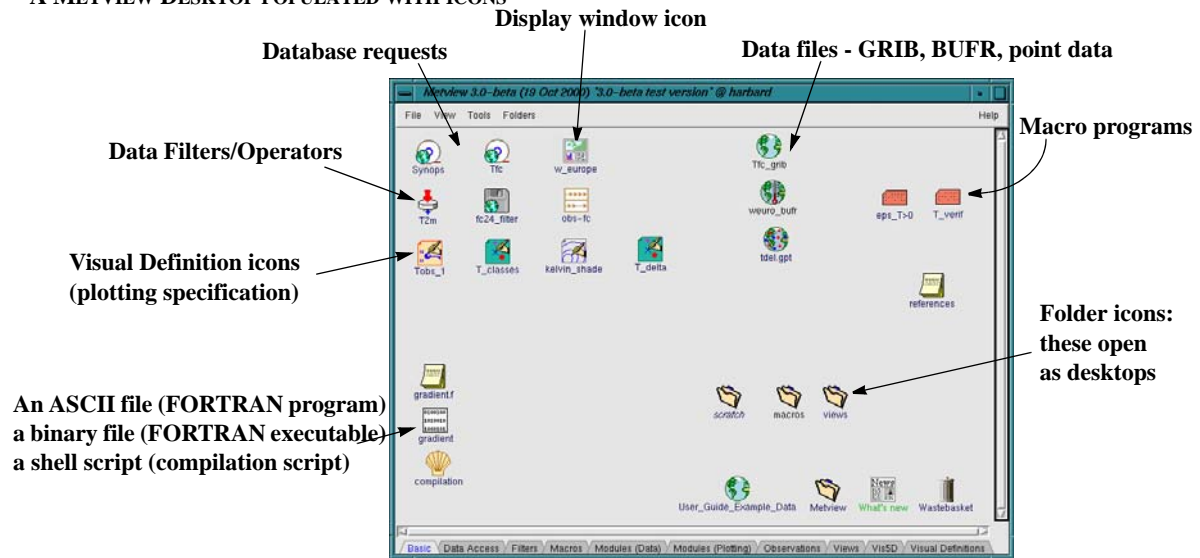


Figure I-1 : An example of a Metview desktop. Icons are used to represent all your work elements, from data files to shell scripts, database requests, executables, macro programs and other desktops

This resulting wide variety of icons is organised in desktop-like interfaces which we designate by **Metview Desktops** - an example is shown in Figure I-1. Physically, these are simply directories and sub-directories. In keeping with the above rule, each desktop when closed is represented by a (folder) icon on the parent interface. Desktops provide all the required functionality to store and manage/handle icons, and you can have as many as you need.

Using this Desktop-Icon architecture Metview goes a long way to enable you to consolidate and structure your meteorological data processing and visualisation work entirely within its environment. Command line exclusivists are also catered for provided they code all their tasks in Metview macro language and run Metview in batch mode.

Starting Metview

Metview is started by running a script which launches the Metview Desktop, an example of which can be seen in Figure I-2. Simply type the following at your UNIX command prompt :

```
% metview
```

Invoked with no options, this script launches the "operational version" of Metview in its default state.

Metview is continuously maintained and updated and any modifications are implemented in a test version so they can be tried prior to release of the new version. To run the *test version* of Metview type :

```
% metview_test
```

To cater for developers and user needs, you can specify a series of options to both scripts :

- To obtain a list of available startup flags and environment variables that are used by Metview, use option -h :

```
% metview -h
```

- To specify a Metview user directory other than your default one (\$HOME/metview) use option -u :

```
% metview -u directory
```

Example :

```
% metview -u /home/other/user/other_metview
```

If the directory does not exist a new one is created. Note that the new directory is created with the standard first-time contents. You could use this to keep two different sets of work completely separated.

- To use a bigger font (useful for demonstrations) :

```
% metview -F
```

- To produce and save information on the output from Metview components (which is blocked by default) use either :

```
% metview -log string
```

or

```
% metview -slog
```

and, optionally,

```
% metview -qlog
```

or

```
% metview -mlog
```

With -log, each Metview component writes its own log file into \$SCRATCH. Log file names are derived by prefixing the Metview component name with *string* and postfixing it with .log; note that *string* is optional and if not provided, default_log is used instead. With -slog, output is written to stdout, i.e. in batch jobs to the normal log and in interactive jobs into the xterm window where Metview was started.

In order to receive full output from Metview for debugging purposes, the environment variable MV_DEBUG_PRINT must be set to 1 before starting Metview. Note that the -log option, by default, is only available at ECMWF. For external users, see the Metview installation pages (<http://www.ecmwf.int/publications/manuals/metview/install/index.html>) for information on how to activate this option. -slog, however, is available on all Metview installations.

Using parameter -qlog ('quiet log') suppresses all log messages apart from error messages. Parameter -mlog ('MARS log') is similar to -qlog except that MARS retrieval messages are printed.

- To execute Metview in the non-interactive *batch mode*, running the macro *macroname* (with or without input arguments following) :

```
% metview -b macroname arg1 arg2 ... argN
```

You can mix some options together - in some circumstances, use of log files can help to debug macros, e.g.

```
% metview -log buglog -b bugmacro
```

The `-b` option must be always the last option specified

- To set your own X11 display :

```
% metview -display
```

THE METVIEW DESKTOP

The Metview Environment

When you start Metview, the first thing you see is the main Metview Desktop as shown in Figure I-2.

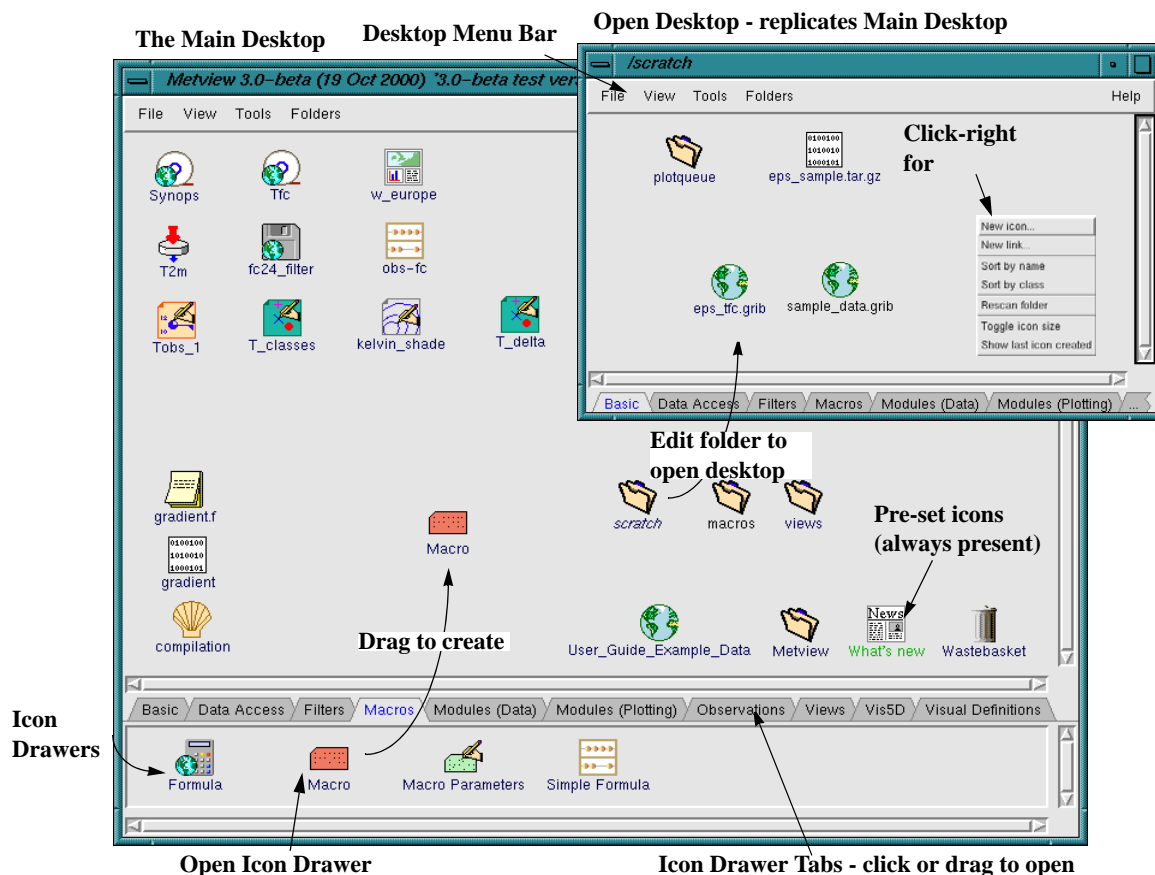


Figure I-2 : The Metview Desktop. The four icons on the lower right of the main desktop are predefined and always appear on start up. Folder and link icons open to reveal a new (replicate) desktop. This example desktop shows a wide variety of user created icons. Users carry out tasks by means of operation on the icons. Icon templates are kept in icon drawers from where they can be dragged to create new instances on a desktop

The obvious feature of the Metview Desktop is the presence of icons, representing your work elements and tools - scraps of text, UNIX shell scripts, observations and model fields, database requests and data filters, plotting specifications, directories and symbolic links, etc. (see "Metview Architecture - Desktops and Icons" on page I- 2). Some icons are present from the very first time you run Metview and they cannot (or should not) be deleted - they are known as **System Icons**, and include the Wastebasket, the Metview (or system) folder and the What's New tool (and possibly the icon of a small data set of model fields for tutorial purposes). All other icons are created by users in the course of their work.

The main Metview Desktop is simply the root of a directory tree. What you see as the main Metview desktop is in fact a UNIX directory called `metview` residing on your root directory (you can specify a different name if you start Metview with option `-u`, see "Starting Metview" on page I-3). All the sub-directories and symbolic links below this main level are represented by Folder icons.

This directory tree structure is known as the **Metview Environment**. If you want to have directories which are not within the Metview environment available as Folder icons in a Metview desktop you have to create symbolic links to them from within the Metview environment (see "Creating Links" on page I-20).

Opening a folder results in a perfect replica of the main desktop with all of its functionality, as seen in Figure I-2. Unlike previous versions of Metview, the main desktop is just another desktop, you can close it and remain with whichever desktop(s) you choose. Throughout this manual it is understood that *desktops* refer to *open* desktops and *folders* refer to *closed* desktops which appear in the parent desktop as Folder icons.

Desktop Functionality

Metview aims to create an interactive work space based on the concepts of desktops and icons. You can conceivably do all your work within the Metview environment given that :

- you can run UNIX shell scripts from a Metview desktop (with Shell icons of course)
- it includes its own Mail tool, able to seamlessly send and receive icons, so you can exchange data and macros with colleagues, report bugs to developers or pose questions to user support
- Folder icons can also represent symbolic links so you can have your scratch space, a data storage directory or in fact any directory for which you have read permission, accessible from any one of your desktops with their contents transparently accessible to your visualisation and processing work

In the course of your work with Metview you will create a possibly large number of new icons which have to be organized in the same way you organise files outside Metview - in a directory structure, i.e. in folders and sub-folders. Icon creation and management are detailed in "Metview Icons" on page I-17.

The desktop functionality is provided by the **desktop menu** and the **desktop bar**, which store desktop wide icon management commands as well as providing user applications and a help system - see "Desktop bar" on page I-7 and "Desktop menu" on page I-9.

The other main component of the Metview desktop are the **desktop icon drawers**, fully customisable storage spaces for both system and user template icons. As indicated in Figure I-2, these are the (main) source of new desktop icons and its customisation allows the user to tailor the desktop to its own work practices and requirements - see an overview in "Desktop icon drawers" on page I-9

Desktop bar

The desktop bar gives access to a series of functions and tools, designed to meet the requirements of user work. These are organised in a number of pull down menus accessible by a click-left from a desktop menu bar. The options on these menus are listed below :

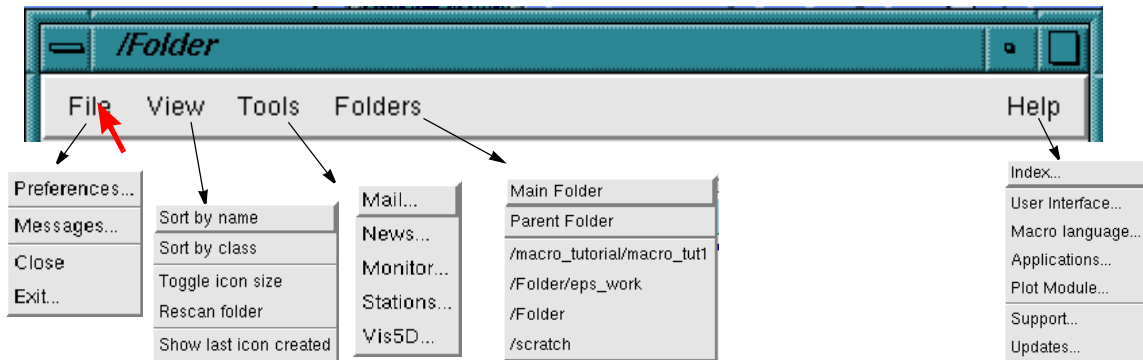


Figure I-3 : The menus on the desktop bar. See text for details

File

Preferences - Launches the General Preferences window. This module specifies the format of the macro language representation of date elements (see "Configuring date formats" in "Converting dates to strings and numbers" on page II- 15), the default printer used by macro programs, the cache retention period and the plotting program to be used

Messages - Opens the session message box - see "The Message Window" on page I- 126.

Close - Closes the desktop. Unavailable on the last open desktop

Exit - Quit Metview

View

This option gives access to a set of commands involved in icon layout, organisation and assignment - the options are the same as those of the Metview desktop menu - see details in "Desktop menu" on page I- 9.

Tools

The **Tools** menu accesses a set of user applications - see "The Metview Tools" on page I- 128

Folders

This option displays a list of the most recently used folders. The top two options are Main and Parent and are always present. Use Main to recover the root desktop window if it has been closed, and use Parent to bring up the parent to the current window. Otherwise select one of the folder names to open its window

Help

The **Help** options give access to the on-line (HTML) versions of the Manuals, and to an e-mailing service dedicated to Metview user support. Full details can be found in "The Metview Help" on page I- 127

Desktop menu

The desktop menu contains commands for managing icons on the desktop - creating new icons, sorting icons, re-sizing icons, updating folder, finding the last icon created. To obtain the desktop menu, simply click-right anywhere on the desktop surface (ensure that no icon is currently selected).

**Click-right on the desktop
to launch the desktop menu**

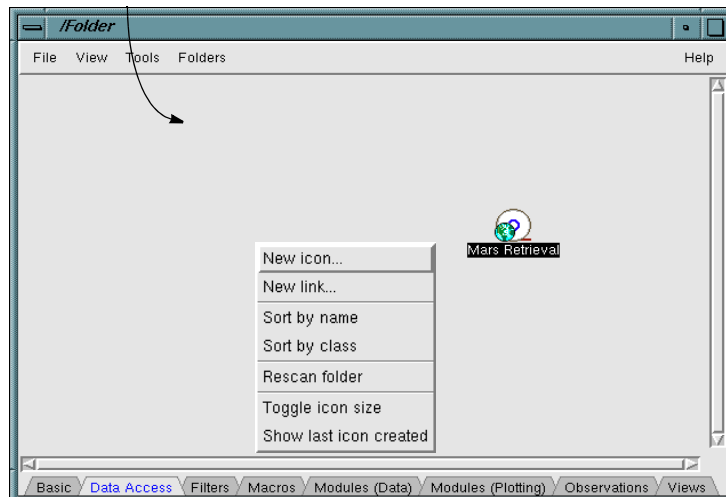


Figure I-4 : The desktop menu. Its options apply to icon desktop management

The desktop menu options are as follows :

New icon - launches a shop window of icons, from where you can create a new one by clicking on the respective picture; see details in "Creating Icons Explicitly" on page I- 18

New link - launches a file browser widget enabling users to select a directory/file to which they can create a symbolic link; see "Creating Links" on page I- 20

Sort by name - sorts the desktop icons by alphabetical order (not case-sensitive)

Sort by class - sorts the desktop icons by type

Rescan folder - refreshes the desktop (in case a new file has been imported/created)

Toggle icon size - toggles between large and small icons

Show last icon created - the last icon created in the current Metview session flashes

All the options except the first two are present in the View menu on the desktop bar (see previous item). See also "Operations on Icons" on page I- 24 for details.

Desktop icon drawers

The lower part of the desktop has a set of tabbed compartments called **icon drawers**. Icon drawers are simply *storage spaces for predefined (default) icons*. Icon drawers store both the default icons provided by the system and those created by the user. The icons inside each drawer are used to cre-

ate new icons on your desktop(s) - see "Creating Icons Explicitly" on page I- 18. Users familiar with previous versions of Metview will recognise these drawers as playing the role of the *Create Buttons*, but with a lot more flexibility, in that users can add their own defaults to the system defaults and also customise the drawers.

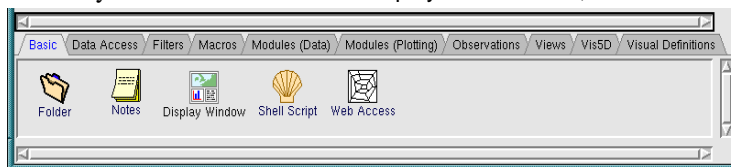
When you start Metview for the first time, desktops have a pre-defined arrangement of icon drawers, each being assigned a pre-defined name (see Figure I-2). This arrangement groups icons according to type, the type being defined by the tasks the icons are primarily involved in.

However this default arrangement of the desktop icon drawers can be modified by the users - drawers can be renamed, deleted and their contents modified at will. This allows users to configure their desktops to match their requirements and preferences - see "Icon Storage - The Desktop Drawers" on page I- 29 for details.

Opening / Closing an icon drawer is done with a left-click on its name tab or by dragging the name tab up or down. Below we list the system icons grouped by the drawers initially provided by the system :

Basic

Icons for basic system work and also the Display Window icon, since it is so frequently used :



Folder - to create new directories

Notes - plain ASCII text editor, use for any general text amount

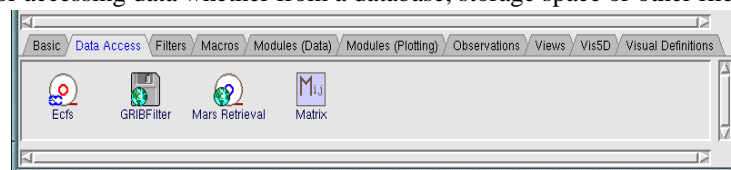
Shell Script - plain ASCII text editor but files are stored as shell scripts

Web Access - launches Netscape with a specified URL

Display Window - designs/launches the visualisation tool

Data Access

Icons for accessing data whether from a database, storage space or other files :



Ecfs - Retrieves files from ECMWF file storage (ECMWF specific)

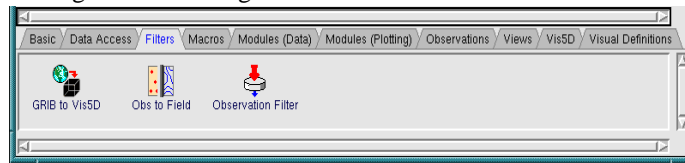
GRIB Filter - Filters model field data from GRIB files or MARS requests

MARS Retrieval - Retrieves model fields or observations from ECMWF archives. Other institutions may have an equivalent icon for their own system

Matrix - Allows access/manipulation of ASCII data matrices

Filters

Icons for filtering and converting data :



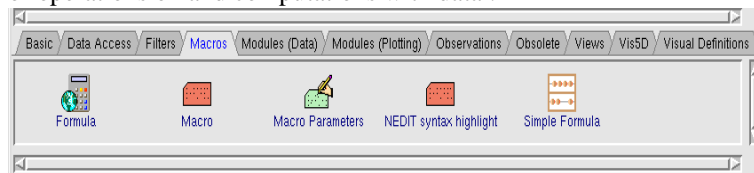
GRIB to Vis5D - converts GRIB data to Vis5D format

Obs to Field - converts irregular point data to a gridded field (GRIB format)

Observation Filter - filters subcategories of observations from BUFR files

Macros

Icons for operations on and computations with data :



Simple Formula - Simple operations and functions on data (geopoints or GRIB)

Formula - Arbitrary operation/function sequences on data (geopoints or GRIB)

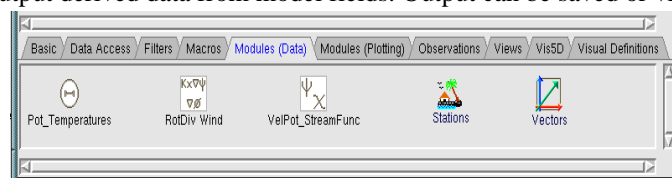
Macro - Macro language editor for Metview macro programs

Macro Parameters - Graphical interface wrapper for macro programs

NEdit Syntax Highlight - Activates syntax highlighting for macro editing using NEdit

Modules (Data)

Icons to output derived data from model fields. Output can be saved or visualised :



Potential Temperatures - Computes potential temperature fields

Rotational / Divergent Wind - Computes rotational or divergent wind vector fields from input of divergence or vorticity

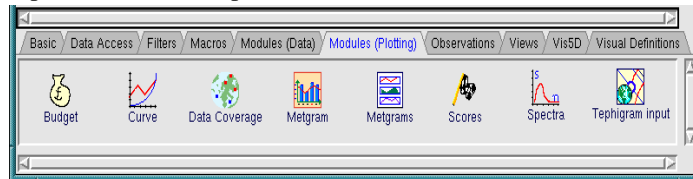
Velocity Potential / Streamfunction - Computes velocity potential or streamfunction fields from input of divergence or vorticity

Stations - Extracts station information from the station database

Vectors - Derives a vector field from two scalar component fields

Modules (Plotting)

Icons to output a visualisation product from model fields :



Budget - Computes and plots several budget quantities

Scores - Generates and plots verification scores (rms only)

Spectra - Generates plots of spectra as a function of Legendre polynomial order

Tephigram input - Retrieves the required data for tephigram plotting

Data Coverage - Plots observations according to user defined quality criteria

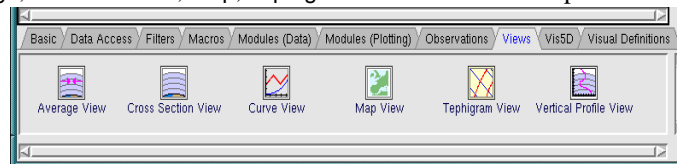
Curve - Plots XY graphs - scatter, line, bar, area plots

Metgram - Plots time series of GRIB or BUFR data for a specified location

Metgrams - Plots the classical Metgram or the EPS Metgram

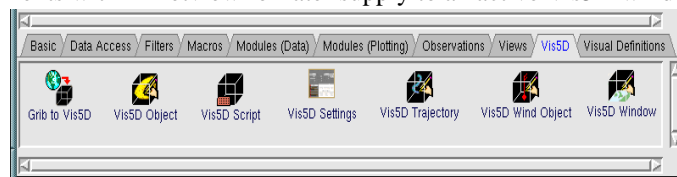
Views

Icons of Views, i.e. visualisation specifications - for instance, the Map View icon specifies area coordinates and projection system for plots of 2D model fields. These icons provide specifications for Average, Cross Section, Map, Tephigram and Vertical Profile plots amongst others.



Vis5D

Icons involved in interfacing Metview with the Vis5D application. These icons allow you to specify Vis5D elements within Metview for later supply to an active Vis5D window.



GRIB to Vis5D - converts GRIB files to Vis5D format

Vis5D Window - specifies a Vis5D window

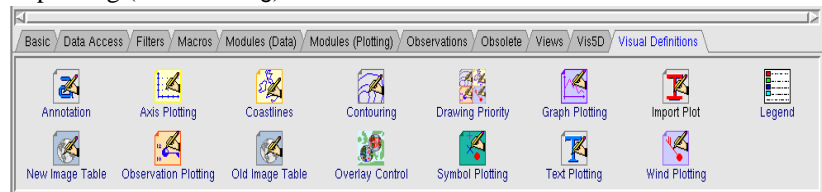
Vis5D Script - a macro-like text editor for Vis5D scripts

Vis5D Object, Vis5D Trajectory, Vis5D Wind Object - these specify Vis5D plotting objects.

Visual Definitions

Icons that specify plotting details of axis systems (Axis Plotting), map frames (Coastlines), contour lines (Contouring), XY graphs (Graph Plotting), satellite image display (Image Table), observation

symbols (Obs Plotting), point markers (Symbol Plotting), text format and location (Text Plotting) and vector plotting (Wind Plotting)



Creating A New Desktop

Metview Desktops correspond to directories and symbolic links. Both are pictured as Folder icons and have exactly the same behaviour and functionality - to allow you to distinguish them, folder-links have their name in italics in contrast to folder-directories.

To create a new desktop, you create a new Folder icon. You can create a new Folder icon in one of two ways :

- drag a Folder icon template from the Basic desktop icon drawer
- use New Icon option from the desktop menu and click the Folder button

These procedures are exactly the same as those used to create any other icon (see "Creating Icons Explicitly" on page I- 18). The result is an icon named Folder, which you should rename to something more meaningful (see renaming in Figure I-6).

To create a folder-link follow the procedure outlined in "Creating Links" on page I- 20. The contents of the folder-link will be iconified automatically by Metview (see "Creating Icons Implicitly" on page I- 17) and they will be transparently accessible to the user.

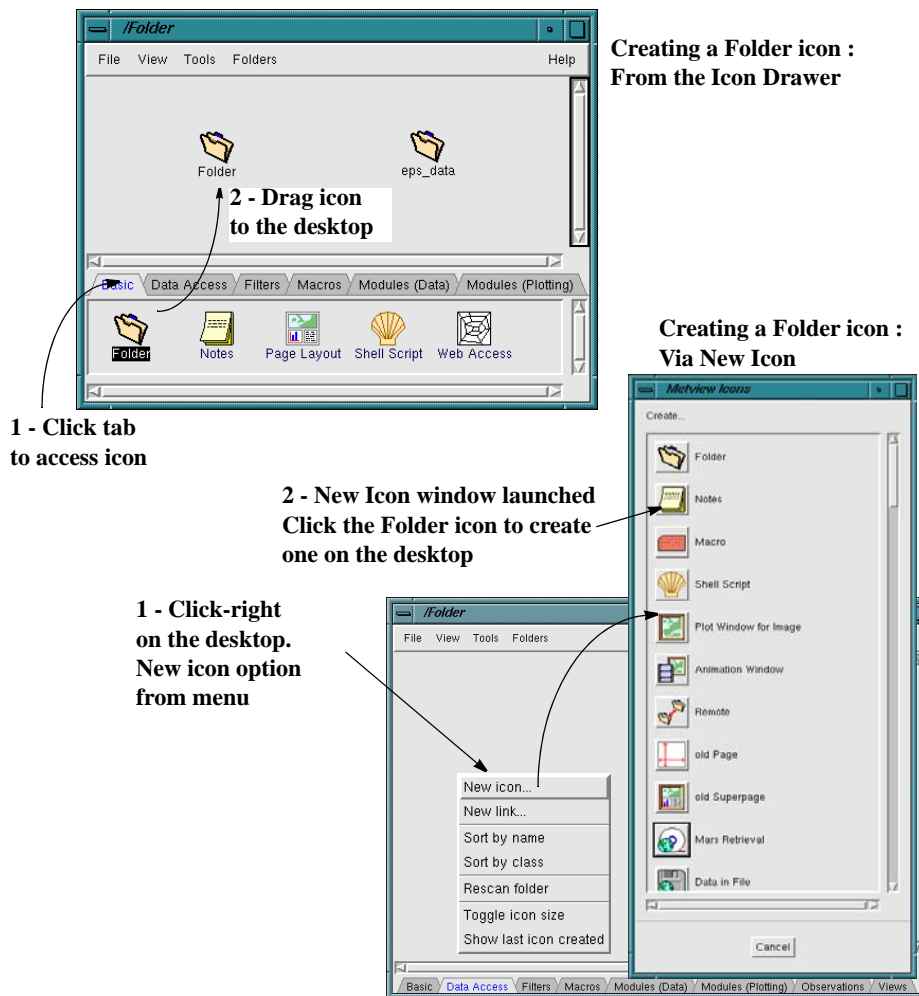


Figure I-5 : Creating Metview folder icons from the icon drawer (above) and via the New Icon option in the desktop menu (below). Folders open as independent desktops, but are otherwise managed like other icons.

Managing Desktops

Metview desktops, whether directories or symbolic links, are Folder icons and as such are managed like other icons - see Figure I-6 for examples :

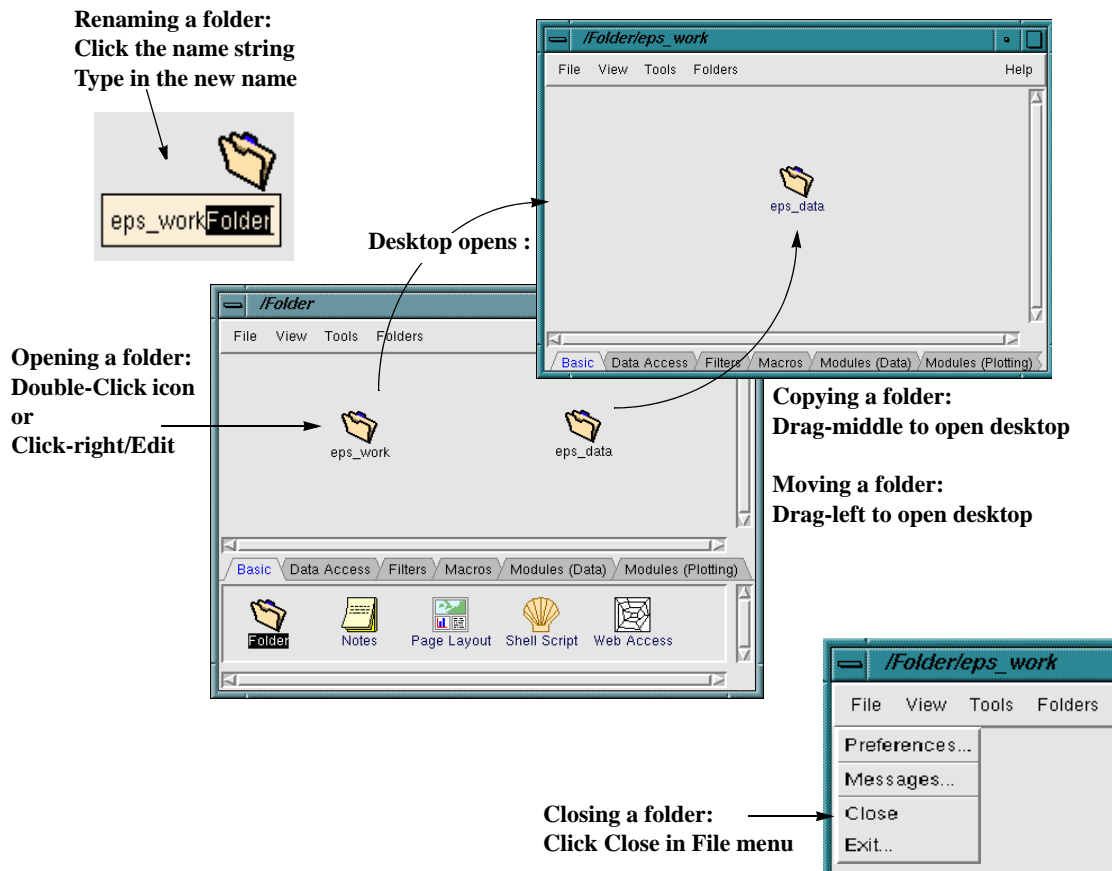


Figure I-6 : Managing Metview desktops/folders. Folders open as independent desktops, but are managed like other icons. Some of the actions pictured are also available from the folder icon menu

- **Rename** a desktop - click once on the folder name string to enable the editing, type in the required new name and Enter to apply
- **Open** a desktop - double-click its icon or click-right and select **Edit** from the icon menu (see "Icon menus" on page I- 23). If you have recently used the folder you want to open, select its name from the Folders menu in the desktop menu bar
- **Move** a desktop to another desktop - *drag-left* the folder icon to the other desktop
- **Copy** a desktop (and all of its contents) to another desktop - *drag-center* the folder icon to the other desktop
- **Close** a desktop - click **Close** option on the **File** menu of desktop menu bar
- **Delete** a desktop (and all of its contents) - click-right and select **Delete** from the icon menu (see "Icon menus" on page I- 23); alternatively drag the icon to the Wastebasket

Closing Desktops and Exiting Metview

You can close each desktop independently using option **Close** on the **File** menu of the desktop menu bar. In effect, you can close the main desktop and work only on whichever desktop you want. You

can always bring back the main desktop (you may have to in order to access folders in other branches of the metview directory tree) with option **Main** on the **Folders** menu of the desktop menu bar.

When there is only a single desktop left, the **Close** option becomes unavailable (greyed out) and you can only use the **Exit** option - this exits Metview completely, closing all open desktops.

The desktops that were open when you exit will open again when you next start Metview. This is useful when you are carrying out work on the same desktop(s) over a period of time and don't want to re-open them at the start of every session (in particular when they are several levels deep).

METVIEW ICONS

Introduction

Metview is described as *icon-based* - "Everything in Metview is an icon". Data files, macro programs, shell scripts, text notes, database requests, applications, directories, visual definitions (contours, wind arrows, observation symbols, ...), everything appears in the Metview desktops as icons. Since everything in Metview is an icon, it follows that - "Every Metview task is a sequence of operations on icons".

Hence Metview provides a great deal of functionality to enable users to handle icons - create, organise, move, copy, rename and delete - and to operate on the icons. This section covers what you need to know about working with Metview icons under the following topics :

- **Creating icons** - Metview icons are created by the user directly (*explicit icon creation*) or automatically assigned to files of recognised types (*implicit icon creation*) respectively - see "Creating Icons Explicitly" on page I- 18 and "Creating Icons Implicitly" on page I- 17
- **Operations on icons** - since any task in Metview is a sequence of operations on icons, you need to know which operations are valid for which icon type and what their outcomes are - see "Operations on Icons" on page I- 24.
- **Managing icons** - here you will find how to handle your increasing number of icons - see "Operations on Icons" on page I- 24

Creating Icons Implicitly

Implicit icon creation is defined as - *the automatic assignment of an icon to a file new to the Metview environment* (see "The Metview Environment" on page I- 6). You can bring/create new files within the Metview environment when :

- you copy or move a file from its original location to somewhere within the Metview environment
- you create a new file within the Metview Environment by non-Metview means, e.g. when you compile a FORTRAN program into an executable in the terminal window
- you create a file within the Metview Environment by means of a Metview procedure, e.g. when you save a MARS Request icon to produce a GRIB file, when you write numeric/graphic output from a macro program to an ASCII/PS file, etc,
- you create a symbolic link within the Metview environment to a directory containing files and you open this folder-link

By generating/importing a file you *implicitly create* an icon for it, as Metview analyses the data format of the newly generated/imported file and *automatically assigns* an icon to it. Files of recognised data formats which are assigned specific icons, include :

- GRIB (editions 1 and 2) data
- BUFR data

- ASCII data matrix (with a specific header)
- Geopoints data (Metview format for irregular point data)
- NetCDF data
- PS, JPEG and PNG

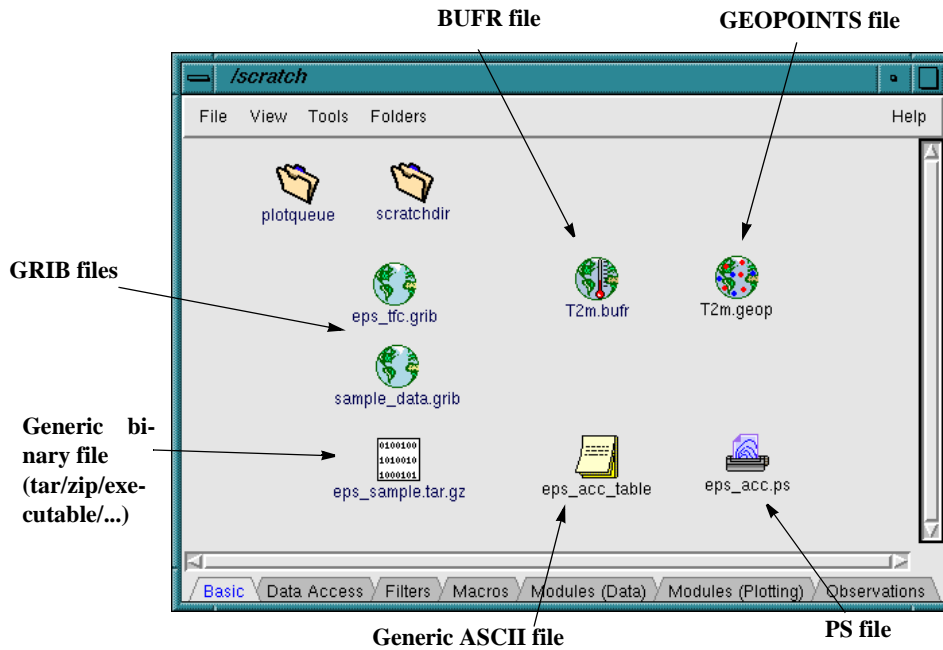


Figure I-7 : Example of icons created implicitly : the files they correspond to are either copied into the Metview system or generated by a macro or an action on another icon. The icons are assigned automatically and cannot be created by the user

If the *data format* is not recognised (i.e. it is not one of the formats above) it is classified according to the *file format* as one of :

- ASCII generic
- Binary generic (all those not recognised as ASCII)

and assigned the respective generic ASCII or generic binary icon.

Since icons of files with recognised data formats and icons of generic binary files can only be created implicitly, their icons are not present in the icon drawers. The automatic iconification of files is usually quite fast since Metview scans and updates its environment frequently. In case you need to speed up the process (unlikely) or to confirm whether a file has been created in a given folder (more likely), use **Rescan Folder** from the desktop menu to reveal the newly created icon immediately. If no new icon appears the file you are looking for isn't there.

Creating Icons Explicitly

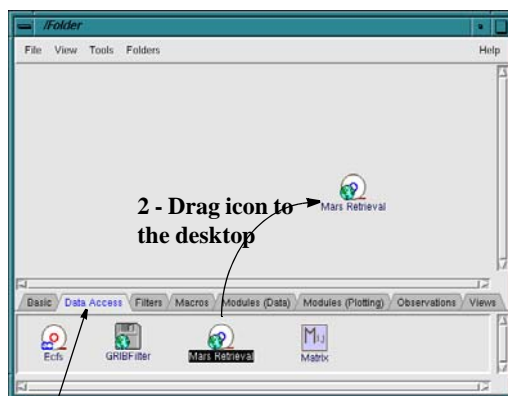
Explicit creation of icons is the creation of icons by a user. This is how most icons are created. You create an icon explicitly in three steps of **Get/Edit/Save** :

Getting the icon

To create an instance of the required icon on your desktop, either :

- Open the icon drawer that contains the icon by clicking its name tab and drag the icon from the icon drawer to the destination desktop.
- Click right on the desktop surface and select option **New Icon** - this launches a "shop window" of icons, where you select the one you need to create.

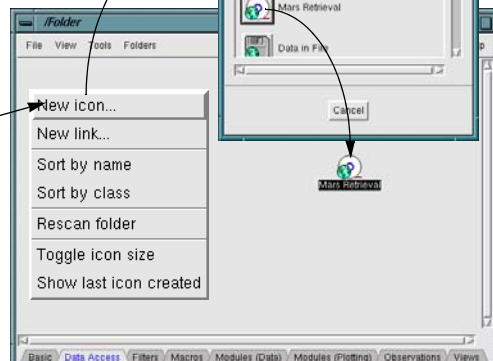
Through Icon Drawer



1 - Click icon drawer tab to access the icon

Through New Icon

1 - Click-right on the desktop. New icon option from menu



2 - New Icon window is launched. Click on the required icon to save it to the desktop

Figure I-8 : Getting an icon (MARS Retrieval) onto your desktop. Getting an icon from the icon drawer is more flexible since you can have your own templates there. The NewIcon window always provides the system default icon.

This creates a copy of the icon on your desktop. If you got it from the icon drawer, the icon contents are those of the template icon you chose to copy. If you got it from the **New Icon** window, the icon contents are those of the original icon template provided at installation time. Either way, unless the template already satisfies your requirements, you have to modify the icon contents - for this you need to **Edit** the icon (next).

Editing the icon

To edit the icon, click-right on the icon and select option **Edit** from the icon menu to launch the icon editor. The icon editor is an interactive window with the functionality to specify the required icon input, in most cases a (possibly long) list of alphanumeric parameters.

For details of working with the icon editor see "Icon Editors" on page I- 34.

Saving the icon

Once you have finished specifying the required input, click the Apply button of the editor window to save the icon contents and exit the editor. Once the process is finished, you can manage the icon - renaming, copying, moving, deleting - according to requirements : see "Operations on Icons" on page I- 24 for details.

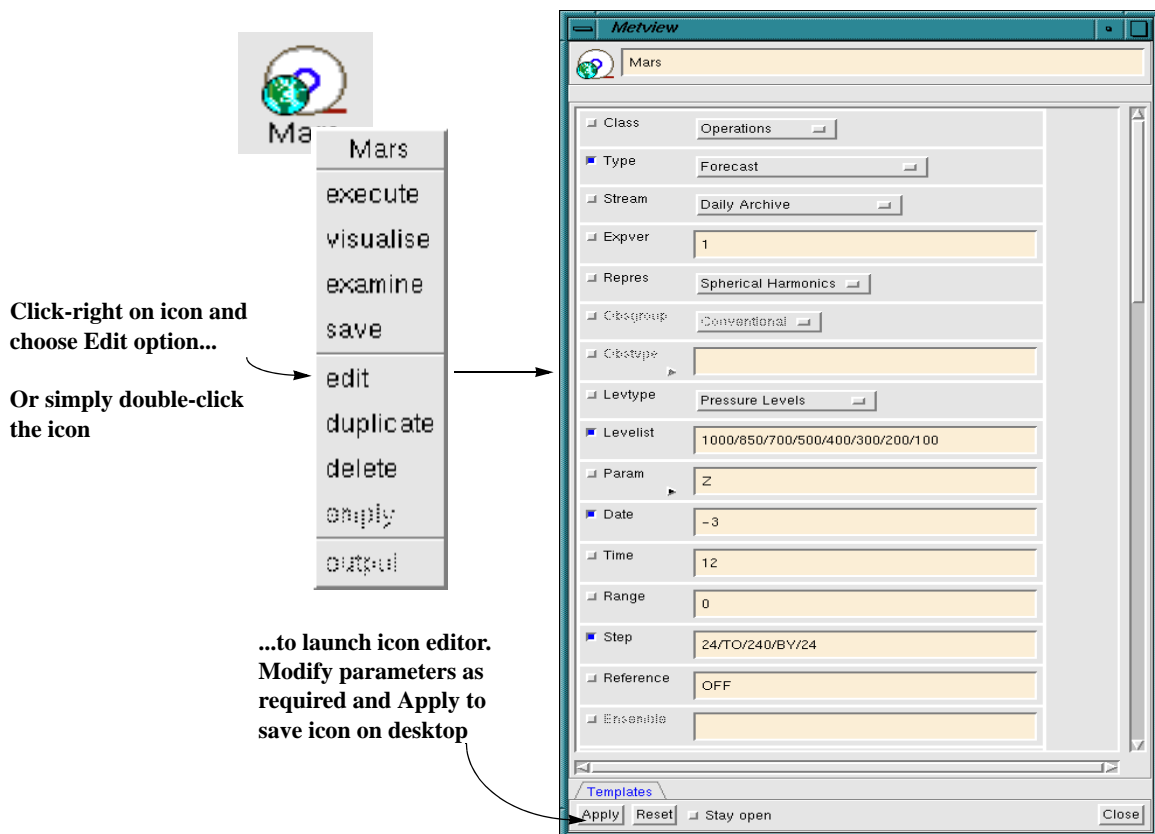


Figure I-9 : Editing and saving an icon. You can double-click it instead of using the icon menu to launch the icon editor. Once you are through modifying the input list, Apply to save the icon

Creating Links

There may be situations when you want access to a file or directory without having to copy them into the Metview environment. This is simply solved using symbolic links in that Metview recognises them and assigns to a symbolic link the icon of the item it targets. Icons of symbolic links are identical to normal icons except for the name which is written in italics.

This is particularly useful when using links to directories - you can simply create within the Metview environment, a link to a data storage directory or to your scratch space or to a colleague's directory, in fact to any directory to which you have read permission. The important feature is that

the contents of the folder/link are iconified and therefore transparently accessible to your work. The rules for iconification are those presented in "Creating Icons Implicitly" on page I- 17.

It is recommended that you create in a Metview desktop, folder-links to your scratch directory and others where you may keep large data files. Links to single large data files can also be useful, though far less used than links to directories.

To create a link in Metview use one of the :

- create a symbolic link in the UNIX command line (ln -s target link). The symbolic link will be automatically recognised by Metview and assigned a proper icon (e.g. Folder icon) with name in italics
- use option New Link on the desktop menu as shown in Figure I-10

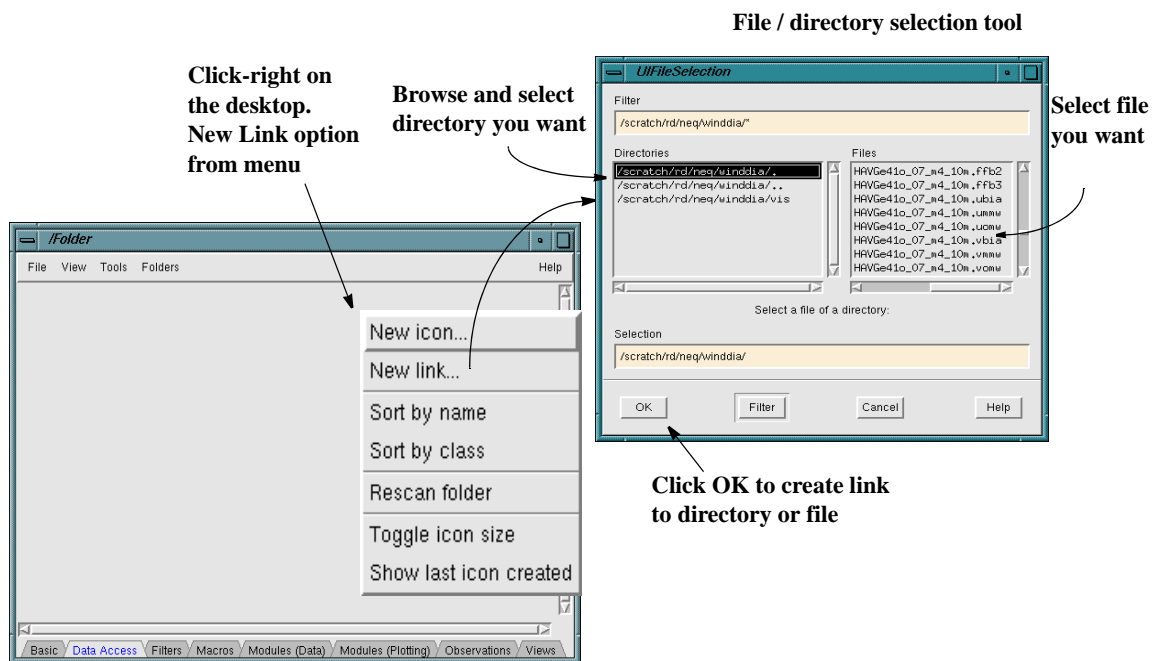


Figure I-10 : Creating a Metview folder link via the New Link option in the desktop menu. Alternatively create a symbolic link in a terminal window, which will be automatically recognised by Metview.

Icon Selection and Icon Menus

To handle icons, whether to operate on them (e.g. to Visualise a data icon) or to manage them (e.g. to move a group of icons to another folder), you have to *Select* them first. When icons are selected you can bring up menus with a choice of operations or apply some form of mouse action to them (e.g. moving them by dragging).

You can select any number of icons irrespective of their positioning on the desktop. Selected icons have the foreground and background colours of its name reversed to confirm selection.

Single selection and the icon menu

To *Select* a single icon (single selection) :

- *Click-left* on the icon
- or
- Type the first letter of the icon name (*not* case sensitive). If several icons share the same initial, repeat until the required one is selected

Once the *icon is selected*, you can bring up the **icon menu** by clicking-right anywhere on the desktop - see Figure I-11. The icon menu provides users with a number of icon operations to choose from (see "Operations on Icons" on page I- 24).

You can obtain the icon menu without explicitly selecting the icon first by right clicking directly on the icon.

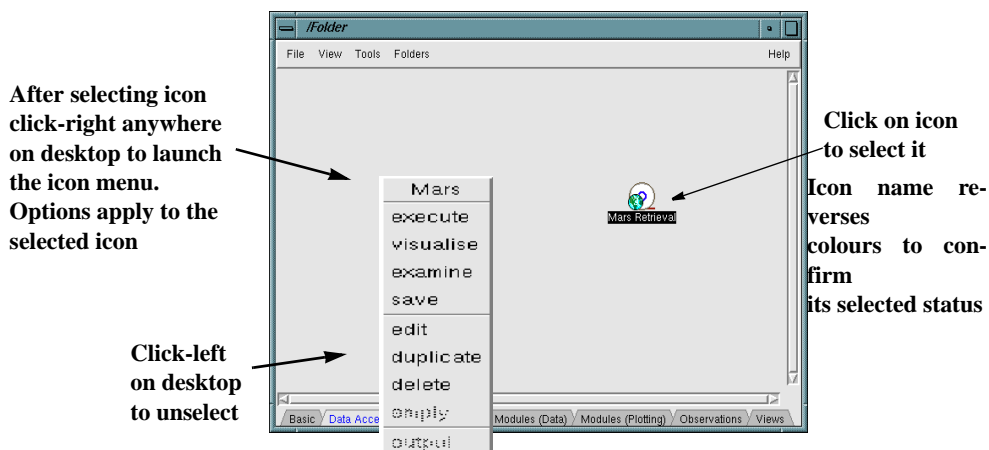


Figure I-11 : Selecting a single icon. Once icon is selected, a click-right on the desktop brings up the Icon Menu. Actions will apply to the selected icon

Multiple selection and the current selection menu

To *Select* more than one icon (*multiple selection*) carry out one of the following :

- *Shift-click* on the icons to be selected. Use this when the icons to be selected are scattered around the desktop
- *Drag* across a group of icons, drawing a rectangular frame (see Figure I-12). When you release the mouse button the selection is made. Use this when the icons to be selected are grouped together on the desktop
- Type **Ctrl-a**, to select all the icons in a desktop

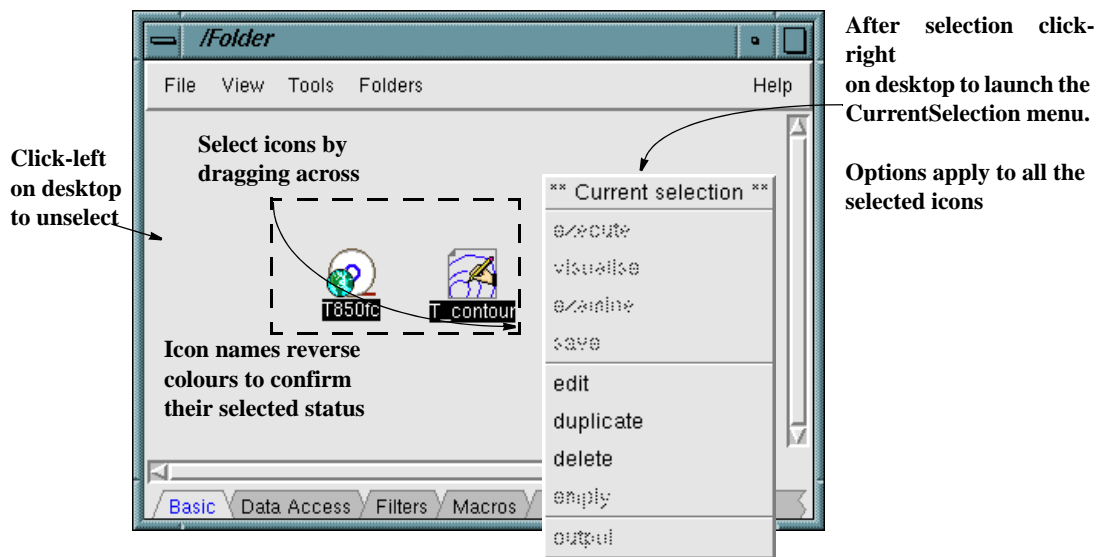


Figure I-12 : Selecting multiple icons by drag-left. Non-contiguous icons can be selected by Shift-clicking them. Once a number of icons is selected a click-right on the desktop brings up the Current Selection menu. Actions will apply to all selected icons

Once a number of icons are selected, a click-right anywhere on the desktop brings you the **current selection menu** - see Figure I-12. This menu also provides icon operations to choose from, but which ones are available depends on the type of icons which have been selected (see "Icon menus" below).

To *Unselect* icons (one or more) :

- *Click-left* on the desktop

Icon menus

Icon menus are available from a click-right on the desktop surface, provided one or more icons are selected. If a single icon is selected you get an icon menu, if multiple icons are selected you get a current selection menu (see above).

The icon and current selection menus list the operations you can apply to an icon or group of icons. The operations applicable to icons in Metview are (see details in "Operations on Icons" on page I-24) :

Execute, Visualise, Examine, Save, Edit, Duplicate, Delete, Empty, Output

As shown in Figure I-13 not all operations are available for all icon types - e.g. you can't *Visualise* a Folder icon or *Execute* a Contour icon; unavailable operations appear in the menu in grey rather than black lettering.

When you make a multiple selection including icons of different types, the current selection menu will only include those operations which are common to all selected icons, as shown in Figure I-13. Operating on a multiple selection of icons allows users to *Duplicate* or *Delete* a set of icons in one go,

launch a visualisation of a set of data retrieval icons or running several database requests simultaneously.

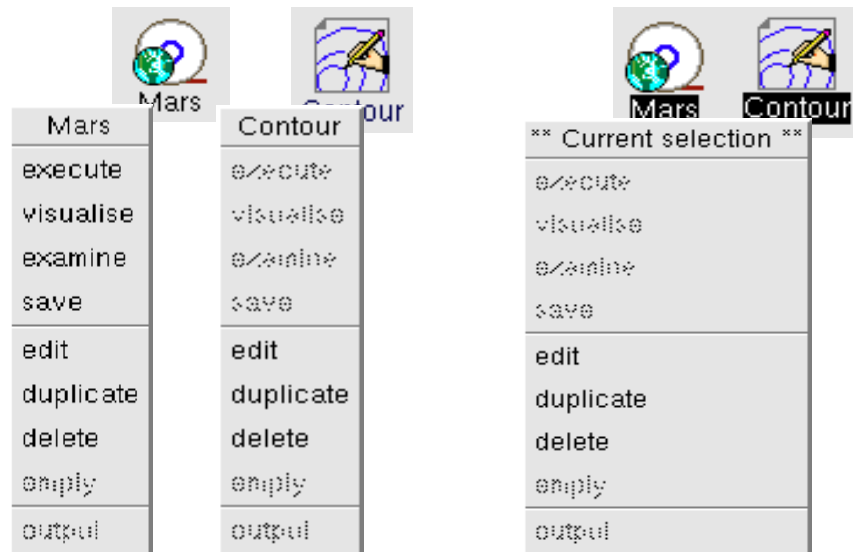


Figure I-13 : Icon menus for two icons of different types and for the current selection of both icons. See text for details.

Operations on Icons

Any task in Metview is carried out as a sequence of operations on icons. The operations applicable to icons in Metview are :

Execute, Visualise, Examine, Save, Edit, Duplicate, Delete, Empty, Output

They include operations involved in icon management - *Edit, Duplicate, Delete, Empty* - and those involved in actual computation/visualisation tasks - *Execute, Visualise, Examine, Save, Output*. These operations are available as options in the **icon menu** or in the **current selection menu** - see "Icon Selection and Icon Menus" on page I- 21.

The *Execute, Visualise, Examine* and *Save* operations are not available for all user icons and lead to different outcomes depending on which icon they are applied to - see details below and in the end sections of the icon descriptions that make up Vol. III- Icon Reference.

The *Edit, Duplicate, Delete* and *Output* operations are available for *all* user icons and always lead to the same result, whichever icon they are applied to :

- *Edit* - opens the icon editor window
- *Duplicate* - creates a copy of the icon
- *Delete* - moves the icon to the Wastebasket
- *Output* - this option is greyed out until a system message is generated; otherwise it opens the message window (see "The Message Window" on page I- 126)

The *Empty* operation applies only to the Wastebasket icon - it results in the permanent deletion of the Wastebasket contents - this can not be undone.

Execute

Execute operation is available only for data icons, i.e. those that can return data - e.g. MARS Retrieval, Formula, GRIB Filter - the Macro type icons and the Display Window icon. Non-executable icons specify visualisation characteristics - e.g. view or visual definition icons.

Execute on a data icon retrieves the data and stores it in cache - nothing visible to a user takes place. *Execute* on a Macro icon carries out the macro instructions, irrespective of its output being data or not. *Execute* on a Display Window icon launches the display window where you can then drop data and visual definition icons.

Visualise

Visualise operation is available for data icons, image icons, View icons and the Display Window icon. Clearly, *Visualise* does not apply to visual definition icons, since these specify how to plot, not what to plot and hence can't be visualised on their own - the exception is the Coastline icon since a meaningful visualisation (over a default Map View) can be achieved without recourse to data.

Visualise on a data icon retrieves the data, places it in cache and carries out its visualisation under the default specification; hence, the *Visualise* operation can be said to include the *Execute* operation. *Visualise* on an image icon (PostScript, JPEG or PNG) displays the image in the default viewer for that file type. The default image viewers can be changed by setting the environment variables METVIEW_PS_VIEWER, METVIEW_JPEG_VIEWER and METVIEW_PNG_VIEWER. *Visualise* on a Display Window icon launches the display window where you can then drop data and visual definition icons.

Examine

Examine operation is available only for data icons. *Examine* provides information about the data returned by /contained in the icon. The information provided depends on the icon it applies to :

Examine on GRIB data icons - GRIB data icons include both the icons of GRIB data files and the icons that return GRIB data, such as MARS Retrieval retrieving model fields from the database, GRIBFilter, ObstoField , etc.,

The outcome of the *Examine* operation on these icons is the launch of a window where you can inspect the headers of the individual GRIB files that make up the data file or that would be retrieved following execution of the icon. For details see "Examining GRIB Data" on page III- 31.

Examine on BUFR data icons - BUFR data icons include both the icons of BUFR data files and the icons that return BUFR data, such as MARS Retrieval retrieving observations from the database (see "Examining GRIB Data" on page III- 31), or Obs Filter icon set to return BUFR output (rather than geopoints) - see "Data Formats" on page III- 13.

The outcome of the *Examine* operation on these icons is the launch of the ObsExam application, an external (non-Metview) application - this applies to ECMWF, other institutions may have different implementations for this operation.

Save

Save operation is only available for data icons which retrieve data from a database, filter data out of a data file or database request or generate a data file themselves (e.g. a Macro program). The operation saves the data returned by the icon on disk, which can be GRIB, BUFR or Geopoints - see for instance "Saving MARS Retrieval Output (as Files)" on page III- 30.

Edit

Edit opens the icon editor window. On a folder it opens the desktop corresponding to the Folder icon. *Edit* is allowed for read-only icons (pictured with a padlock), but modifications are not saved.

Edit operation is available for all icons and is the default operation, i.e. the operation which is carried out when you double-click an icon. However, there are icons which have no editor associated with them - these are icons representing items such as PS files, GRIB files, BUFR files, NetCDF files, binary executables, etc, i.e. non-ASCII files. The only thing you can do with these files is to rename them.

Full details on working with the icon editors are provided in "Icon Editors" on page I- 34.

Duplicate

Duplicate operation is available for all icons. It creates a copy of the icon named "Copy *N* of *icon name*", where *N* is the number of copies made so far. You may then rename the icon.

Delete

Delete operation is available for all icons. It moves the icon to the Wastebasket folder. Deleted icons are kept there from session to session and are completely removed only when the Wastebasket is emptied (see "Empty" below).

Output

Output operation opens the message window for the icon. This operation only becomes available when a message has been created as the result of some operation on the icon - this could be a system message (e.g. database retrieval information upon executing a MARS Retrieval icon) or console output (e.g. the result of a call to the print() function in a macro program).

Messages are only generated for executable and visualisable icons, so only these will ever have the *Output* operation available. *Output* opens the message window for user consultation and/or saving - see "The output action" on page I- 27.

Empty

Empty is only available for the Wastebasket icon and as the name implies, it deletes the icons from within this folder - icons removed from the Wastebasket are permanently removed and this cannot be undone.

Icon Feedback

Overview

When you carry out an icon operation you need feedback from Metview so you can gauge the progress or outcome of the operation. The operation may also have originated system messages, such as error warnings or data retrieval reports.

Metview provides two types of feedback functionality :

- the icon visual report of the operation status
- the icon message window launched by the output action (icon menu)

Icon status

Following the operations *Execute*, *Visualise*, *Save* and *Examine*, the icon name changes colour to report the status / outcome of the operation. Each colour corresponds to a different icon operation status as follows :

Black	No operation executed since last save (default)
Red	Operation failed, e.g. due to invalid input parameters
Orange	Operation in progress, e.g. waiting for data retrieval from database
Green	Operation successfully executed

Once an operation has ended, the colour of the icon name (green or red) stays until you edit and save the icon or until you re-start Metview.

The output action

When you operate on an icon some text output is nearly always generated. This may consist of one (or more) of the following :

- Error messages, e.g. from a failed data retrieval
- Information messages , e.g. on the progress of a data retrieval task
- Any text output generated by an icon, most frequently a macro program console output generated by the macro function print()

All this icon output is accessible to users by means of an **Icon Message Window** which records the system feedback that refers to this icon alone (error and information messages) as well as any console output from the icon (for Macro-class icons).

The icon message window is simply a pop up where messages and text output are displayed. Users access this message window via the icon *Output* operation :

- click-right on icon
- select Output to open the icon message window

Users can cut and paste to other destinations, e.g. a Notes icon.

Note that there is also a system wide message window to where all text output generated during a Metview session is channelled - see details in "The Message Window" on page I- 126.

Handling Icons on the Desktop

Here we provide details on how to handle your icons on the desktop. This includes operations applicable to all icons in a desktop and operations which are only applied to the icons currently selected.

Moving icons

First select the icons to be moved (see "Icon Selection and Icon Menus" on page I- 21). To reposition icons within the same desktop, simply *drag* the selected icons to the required position. To move the selected icons to another folder, *drag* them to the destination's open desktop or to *exactly* over the destination's folder icon - the destination folder icon will flash upon a successful move. To drag a bunch of icons, it is enough to drag one of them.

Copying icons

First select the icons to be copied (see "Icon Selection and Icon Menus" on page I- 21). To copy selected icons *within the same desktop*, either drag-middle the selected icons or choose **Duplicate** from the icon / current selection menu - all selected icons will be duplicated. Duplicate icons are named "Copy *N* of *icon name*", where *N* is the number of copies on the desktop. You may then rename the icon.

To create a copy of the selected icons *in another desktop*, open the destination desktop and drag-middle the selected icons to it. In this case, the icons keep the name unchanged.

Deleting icons

First select the icons to be deleted (see "Icon Selection and Icon Menus" on page I- 21). To *Delete* icons, choose **Delete** from the icon / current selection menu - all selected icons will be deleted. Alternatively, you can move the icon(s) straight into the Wastebasket folder.

Deleted icons are placed in the Wastebasket. They are kept here between Metview sessions. To *Undelete* icons open the Wastebasket folder and drag them out.

If you choose option **Empty** from the Wastebasket icon menu all icons inside the Wastebasket will be permanently removed - *this action cannot be undone*.

Finding icons

As your work progresses, you may end up with a large number of icons on your desktops. If you have trouble finding a given icon, there are two helpful actions you can take :

To find an icon when you know the icon name - type its first letter (not case sensitive); repeat to cycle through the icons starting with the same letter. *Note that the found icon becomes selected*.

To find a newly created icon - choose **Show Last Icon Created** from the desktop menu to make the newly created icon flash. The behaviour of this option is undefined if no new icon has been created in the current Metview session.

Re-arranging icons

You can re-organise the icons in your work space alphabetically or according to class.

To organise alphabetically (not case-sensitive!) do one of the following :

- type Ctrl-s
- click-right on the desktop and select Sort by Name from the desktop menu
- select Sort by Name from the View menu in the desktop bar

To organise by icon class, do one of the following :

- type Ctrl-Shift-s
- click-right on the desktop and select Sort by Class from the desktop menu
- select Sort by Class from the View menu in the desktop bar

Re-sizing icons

You can choose between large icons (icon name written below the icon) or small icons (icon name on the right hand side of icon). To toggle between the two icon styles, do one of the following :

- type Ctrl-w
- select Toggle Icon Size from either the View menu in the desktop bar or from the desktop menu (click-right on desktop)

Icon Storage - The Desktop Drawers

Overview

The default icons present in the desktop icon drawers are one of the two sources for explicit icon creation (the other being the New Icon pop-up window) - see "Creating Icons Explicitly" on page I-18.

When you start Metview for the first time, desktops have a pre-defined arrangement of icon drawers, each being assigned a default name (see Figure I-2). This arrangement groups icons according to type, the type being defined by the tasks the icons are primarily involved in. A major feature of the Metview desktop is that you can change the desktop look by customising the desktop icon drawers - you can :

- rename the drawers
- create new drawers
- delete existing drawers

and you are free to change their icons at will - you can create new icons within the drawer, you can drop icons from your desktop into any drawer and edit the ones currently residing within the drawer.

The customisation facilities give you the freedom to structure your default icons the way you want it. You can organize the icon drawers according to contents - a drawer for specific types of data retrievals, another for display window layouts - or according to work projects - e.g. by keeping all icons relative to a project in a special drawer (see Figure I-14).

This facility is available because the desktop icon drawers are really just directories placed in particular locations within the Metview environment - they are so-called **system directories**, i.e. subdirectories of the Metview System Folder - this is a folder where system settings and default are kept. A desktop icon drawer corresponds to the system directory :

```
$HOME/metview/Metview/Stationery/DrawerName
```

where *DrawerName* is the name of the icon drawer in question. So, icon drawers can be accessed just like any other directory - knowing the path to the drawer folder, you can access the folder and open it as a Metview desktop, though in normal working circumstances you can simply access any drawer folder directly from the drawer itself.

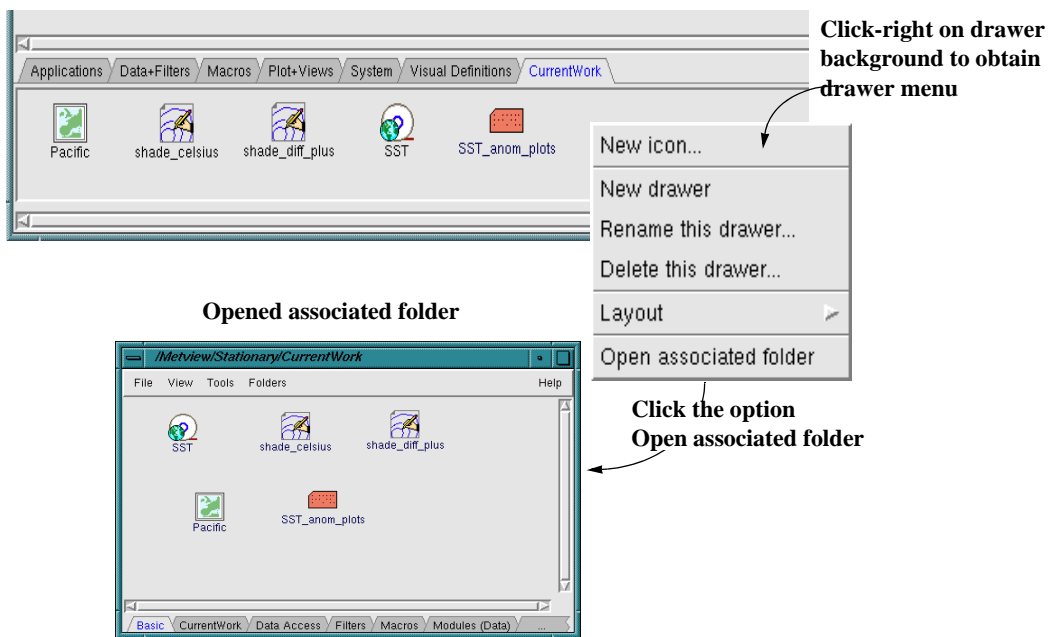
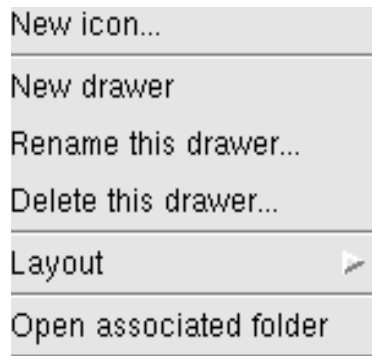


Figure I-14 : Example of a user-customised group of icon drawers. Note the renaming of most drawers, deletion of unnecessary ones, and the creation of a drawer for a user project loaded with pre-specified icons. Also shown is the drawer menu where customisation options can be found and the system folder corresponding to the drawer

Customising the icon drawers

The drawer customisation is done with the *Icon Drawer Menu*, obtained from a click-right on the drawer surface (away from any icon) :



To create a new drawer - click right and select New Drawer from the menu. You can do this from any drawer. The new drawer is created at the end of the existing drawer group and gets the default name Folder. You should rename it to your liking

To rename a drawer - Open the drawer you want to rename, click-right and select Rename This Drawer from the menu. A small window pops up, you type in the new name and click OK to accept

To delete a drawer - Open the drawer you want to delete, click-right and select Delete This Drawer from the menu. The drawer is deleted and you are returned to the first drawer in your arrangement

To open the folder associated with the drawer - Open the drawer whose associated folder you want to open as a desktop, click-right and select the option Open associated drawer as shown in see Figure I-14.

Handling the icons in desktop drawers

Drawer icons are handled in the same way as plain desktop icons, though the only action you can take on the drawer icons is to rename, edit or delete them.

To create a new icon - to create a new icon in a drawer, click-right, select New Icon from the menu to launch the new icon tool and select the required one

To add an icon - to import an icon from a desktop to an icon drawer, simply drag the icon into the drawer

To rename an icon - click-left the icon name string once, type in the new name and enter

To edit an icon - click-right on the icon picture, select Edit to open the icon editor

To delete an icon - click-right on the icon picture, select Delete to delete the icon

To copy/move an icon - to copy an icon from a drawer to another, copy the icon to the desktop, open the destination drawer and copy the icon to there. Alternatively, open the associated folder (see "Overview" on page I- 29 and Figure I-14) of the destination drawer and copy the icon to there. Icon traffic to and from drawers is realised by icon copy, so to move icons you require the same steps but have to delete the copies left behind

Icons Revealed

What you see as an icon in the Metview environment is actually composed of two files - the **dot file** and the **icon file**.

- **dot file** - this file has the name of the icon as seen on the Metview desktop but preceded by a dot ("."). The dot file is always a small and simple ASCII file. The leading dot makes them invisible to the default UNIX file listing command (ls).
- **icon file** - this file has the name of the icon as seen on the Metview desktop. The icon file of icons which can only be created *implicitly* ("Creating Icons Implicitly" on page I- 17) can have a variety of formats - GRIB, BUFR, PS, etc,. The icon file of icons created *explicitly* ("Creating Icons Explicitly" on page I- 18) is always a small ASCII file with a very simple structure

To see this, consider the folder pictured in Figure I-11 (i.e. a directory with a single MARS Retrieval icon). Suppose you prepared the MARS icon to retrieve T850 forecasts and renamed the icon to T850fc. If you use a terminal window to access it and check its contents with :

```
% ls -l -a
```

you get the following directory listing (details may vary):

```
drwxr-x---      3   uid   gr 1024   Oct 11 15:19   ./
drwxr-x---     21   uid   gr 3072   Oct 11 12:44   ../
-rwxr-xr--      1   uid   gr 189    Oct 11 15:28   .T850fc
-rwxr-xr--      1   uid   gr 36     Oct 11 15:19   T850fc
```

This shows the two component (dot and icon) files that compose the desktop icon. They are both ASCII files, so you can type their contents to the console, as such :

```
% cat .T850fc
  USER_INTERFACE,
  ICON_NAME       = T850fc,
  X               = 381,
  Y               = 129,
  NB_OF_TIMERS = 0,
  ICON_CLASS      = RETRIEVE,
  RENAMABLE       = TRUE,
  TEMPORARY       = FALSE
```

As can be seen from its contents, the **dot file** controls the *icon identity and layout* - it stores the name of the icon, defines the class to which it belongs and its location on the desktop (X and Y are window coordinates).

```
% cat T850fc
  RETRIEVE,
  TYPE    = FC,
  LEVELIST = 850,
  PARAM   = T,
  DATE    = -3,
  STEP    = 24/TO/240/BY/24,
  GRID    = 1.5/1.5
```

As can be seen from its contents, the **icon file** controls/stores the icon's list of input parameters - it is a simple ASCII file with the editor input parameters organised according to the syntax exemplified above - users familiar with the MARS application language will recognise it immediately.

If you had a PS file or a GRIB file icon in the same desktop, the directory listing would show you a dot file for this PS or GRIB file, and an icon file which would be the PS or GRIB file itself.

As a rule, editing of these files should be done with the Metview icon editors. In fact, the icon editor allows you to access the text content of the icon file directly, should you need/prefer to do so (see "Editing Icons in ASCII Mode" on page I- 50).

If you have to access these files by external means (e.g. copying to a remote location), bear in mind you may need to handle both the icon and the dot files. Also if setting permissions to allow access to your icons, you must set the correct permissions on both the dot and the icon file.

ICON EDITORS

Overview

This chapter describes the icon editor functionality. Additionally, a complete editor by editor description makes up the last volume in this manual : Vol. III- Icon Reference.

The icon editor is launched by selecting *Edit* from the icon menu, or by double-clicking on the icon (since Edit is the default action). Metview icons require users to specify a variety of input parameters via an interactive **Icon Editor** - examples of these editors are presented in Figure I-15. There are icons which have no editor associated with them and these are non-ASCII data or graphical output files - GRIB, BUFR and NetCDF icons, PS, JPEG and PNG icons, as well as binary executables.

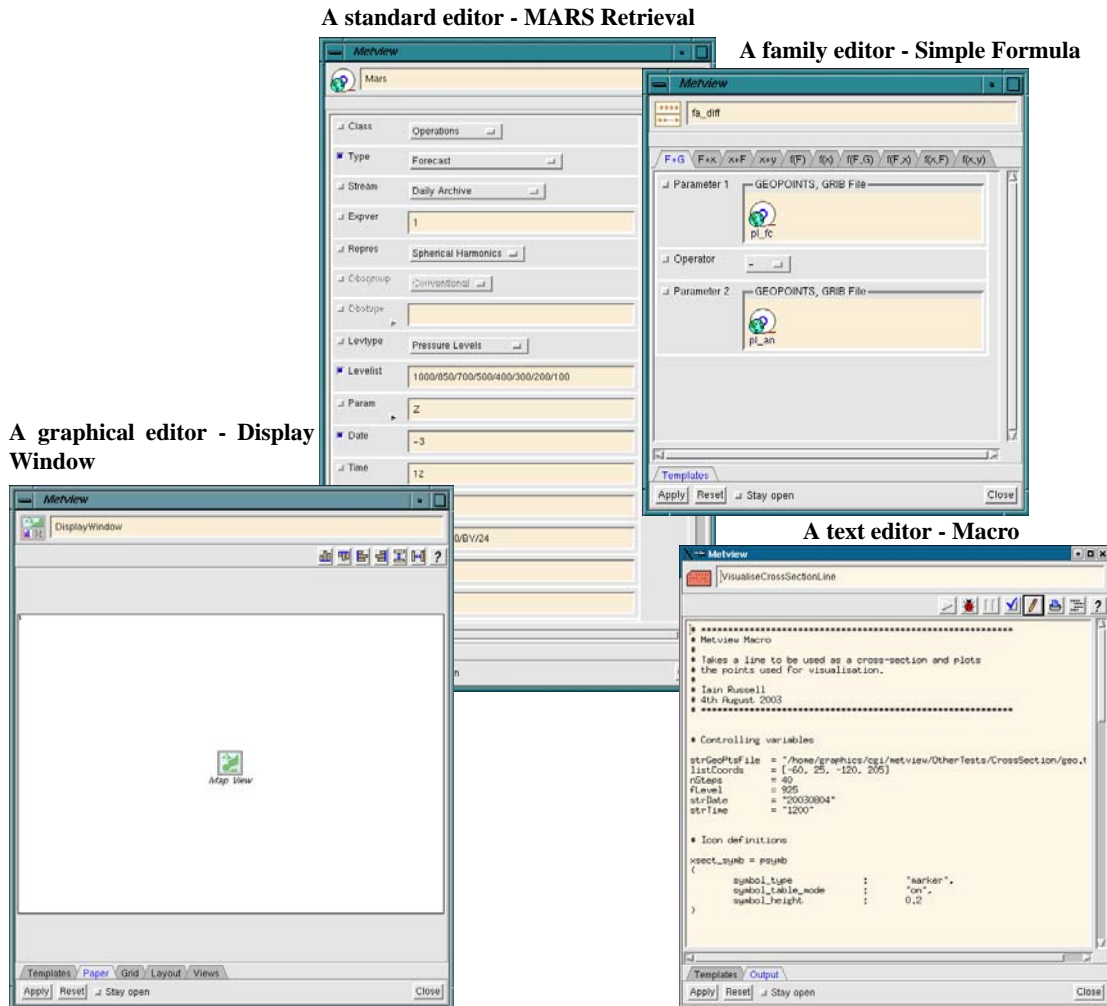


Figure I-15 : Example of the main types of Metview editor - Standard editors (MARS Retrieval), a Family editor (Simple Formula), a plain text editor (Macro) and a graphical interface editor (Display Window)

From the point of view of the input specification, the Metview icon editors can be of three main types :

- plain text editors (Notes, Shell, Formula and Macro)
- purely graphical interfaces (Display Window)
- sequential lists of input parameters (all other icons)

The latter may contain a single set of input parameters (most of them) or a number of separate but related sets of input parameters. In this last case they are known as **Family Editors** an example of which is shown in Figure I-15; each separate set is contained in an independent editor pane, accessed by a tab - examples include the Simple Formula icon (comprising a family of formula and function types) and the Potential Temperature application icon.

You will note that icons can be used as input to other icons and these input icons are known as **embedded icons** (see "Embedded Icons" on page I- 44).

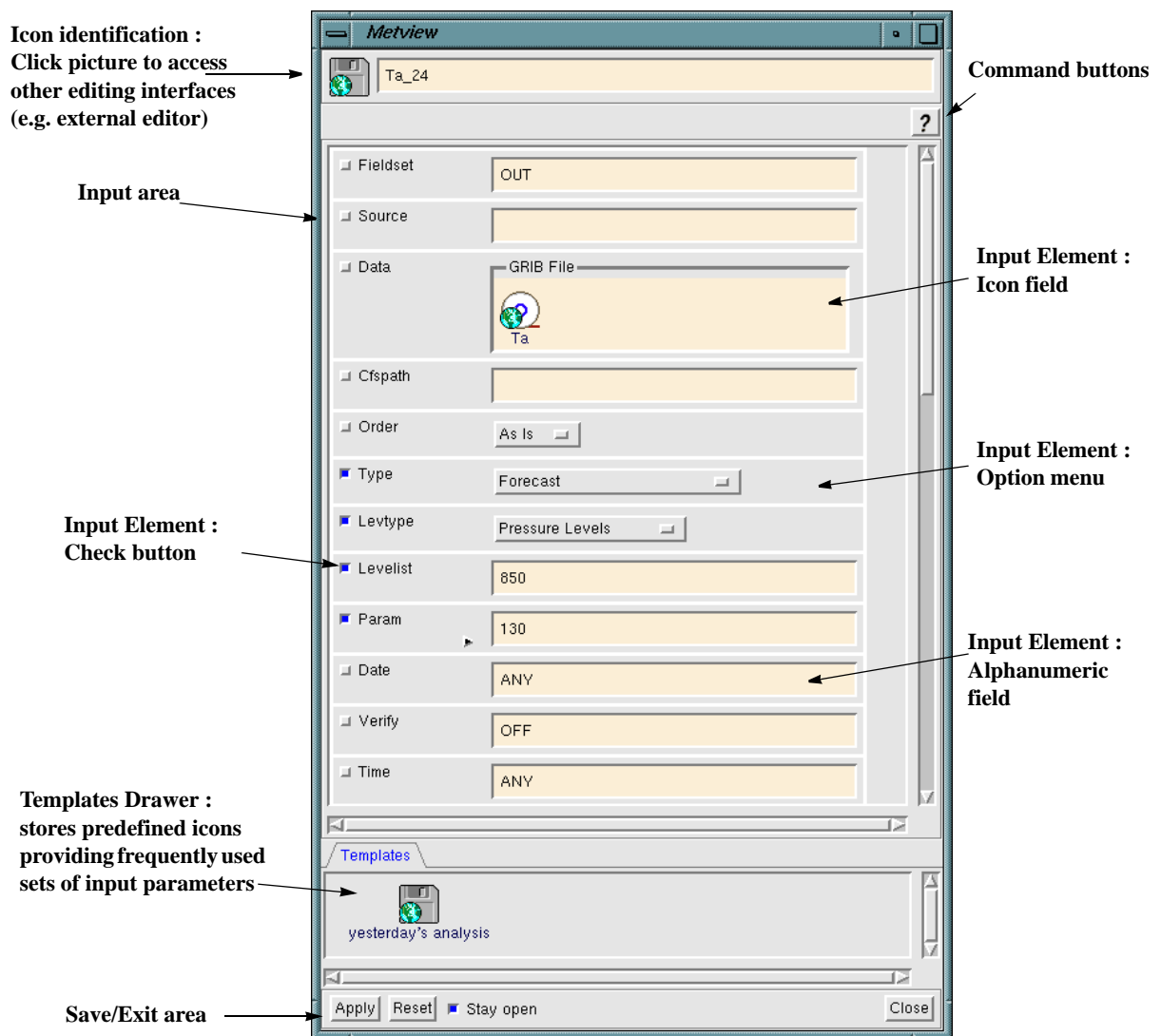


Figure I-16 : A GRIB Filter editor with its main components highlighted. Other icon editors have the same features and general structure.

The Icon Editor Window

A few icon editor windows are pictured in Figure I-15. We consider the icon editor windows structured into a number of different components, highlighted in Figure I-16 and listed below :

Icon identification

At the top of the window users can find an editable text field for the icon name and a picture of the icon. You can always edit the icon name even if you cannot edit the file contents (e.g. binary data files). Clicking the picture of the icon gives access to alternative editing interfaces of the icon content - see "Editing Icons in ASCII Mode" on page I- 50.

Command buttons

Below the icon identification there is a button bar. For most icons it contains only a help button (see "The Metview Help" on page I- 127), but for a couple of other icons, sets of editing buttons are present - in the Macro icon, these include buttons for syntax checking, to run the macro in debug mode, to beautify the code, etc, - see an example in Figure I-15 ; in the Display Window icon there is a set of buttons to control alignment of the visualisation frames.

Input area

The input area makes up most of the editor windows and except for the Display Window editor it is either a plain text window (Macro, Formula, Notes, Shell) or a list of input parameters (all others). The input area lists the icon's **input parameters**.

Content-wise, the input parameters required by icons can be of a wide variety, including :

- data (files or database requests),
- geographical details (area limits and cross section coords, station locations)
- model level lists
- yes/no, on/off or multiple option choices
- dates and time steps
- plotting item (contour line, symbols, arrows, ...) styles
- axis specifications
- text for titles, legends and anotations
- etc...

Format-wise, input parameters can be specified as :

- one item from a list of possible values
- one or several alphanumeric elements (strings and/or numbers)
- other icons
- colours

You specify values for the input parameters through editing tools named **input elements**. Each input parameter is associated with an input element. Input elements are described in detail in "Editor Window Input Elements" on page I- 37 and include :

- option menus and on/off toggles - to select one option from a list
- alphanumeric fields - to enter strings and/or numbers. Note that list elements must be separated with a forward slash (/) character
- sliders - to select a numeric value from a continuous range
- icon fields and associated help icon drawers - to specify other icons as input
- colour lists - to specify a colour
- check buttons - to flag parameters as non-default and to re-set to default

Templates drawer

This tabbed drawer stores predefined icons of the same type which specify standard/frequently-used/hard-to-remember sets of parameters - e.g. a MARS Retrieval predefined for your favourite retrieval specification. To use these templates either :

- drag the icon into the input area of the icon being edited
- use its icon menu to replace or merge the settings of the icon being edited

See "The template icon drawer" in "Template Icons" on page I- 48.

Command drawers

These drawers are only present in the MARS Retrieval and in the Display Window editor. They contain command interfaces - in the MARS Retrieval for the on-line MARS catalogue, in the Display Window editor for the Layout design and implementation work.

Save/Exit area

This area is common to (and identical in) all editors. Buttons for saving, closing, exiting and re-setting reside here.

- **Close** exits the editor window without saving
- **Reset** returns all settings to the values they had when last saved
- **Apply** saves and exits. However, you can check the Stay Open option to save the icon contents keeping the editor window up

Editor Window Input Elements

All Metview icons require input parameters to be specified. Each of these parameters is listed in the icon editor window, associated with an input element, which you use to specify a value (except for the plain text editors). This section shows you how to operate with these input elements.

Dependency between input parameters

You will note that not all input parameters listed in an editor window are available for editing. Those which aren't available have their names and input elements greyed out.

This happens because choosing a given value for an input parameter may make a number of other input parameters not applicable : e.g. in the Contour icon editor if Contour_Line is set to OFF, all the parameters relating to contour line specification (colour, thickness, spacing, etc) no longer apply. The reverse can also happen, when parameters become available only after another parameter is set to a given value. An example is provided in Figure I-19.

Another type of dependency can happen where setting a parameter to a given value, forcibly changes the value of another parameter.

This is quite useful for editing tasks, but only simple dependencies are available. Some icons have complex relationships between input parameters and the editors are not prepared to take all of them into account to decide all the parameters that are available based on a given configuration of parameters already set.

Check buttons

These are present in all editor windows, always preceding each and every parameter name. Every-time you change the parameter from its default value, the check button on its left turns blue - this indicates which values have been set as non-default (see Figure I-17). If you *click-left* on a blue check button, it will turn grey and the value for the corresponding parameter will be reset to the default value.

Text fields

Text fields are the most widespread input tool, since they can accept any alphanumeric combination. Simply type in whichever string or number is required, deleting or modifying the previous contents. Note: *the mouse pointer must be inside the text field*. When you finish entering the new input, hit return - the check button will turn blue, indicating a change from the default. You may not hit return, as the value will still be entered; however, you won't see the check-button turn blue, until you open the editor again after having saved and exited.

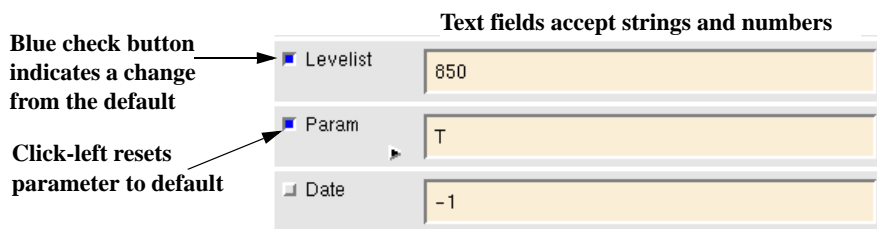


Figure I-17 : Example of text fields and check buttons from a MARS retrieval editor window. Note the coloured check buttons, indicating that the defaults have been changed.

Lists

When a list is required, elements must be separated with a forward slash (/) character.

Selection help lists

Some parameter text fields accept values out of a very large number of different possibilities. For these parameters, the editor provides a drop-down selection help list with an exhaustive listing of all the choices available. The most obvious and frequently used is the drop down selection help list for **Parameter** in the MARS Retrieval icon - it lists all the physical variables which are available for retrieval in the MARS archive.

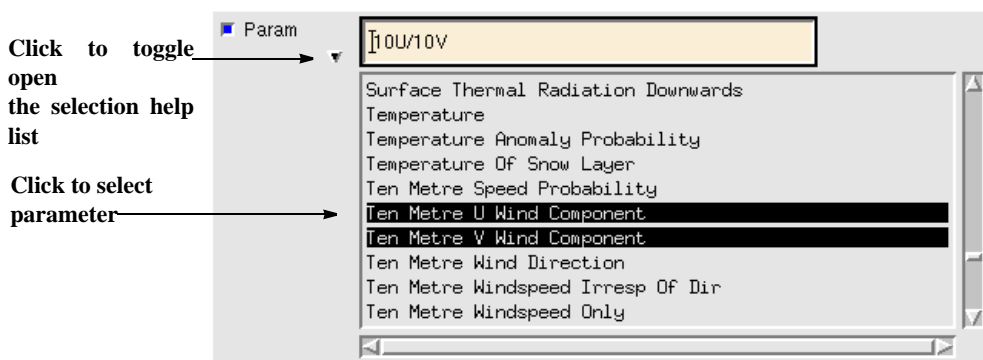


Figure I-18 : Example of a selection help list associated with a text field. Click on the list elements to add them to the text field. Click again to de-select and remove from text field.

Option buttons

Option buttons allow you to pick a choice from several options. The currently selected option is displayed. *Click-left* on the button and then again on the required option (or *drag-left* and release mouse pointer over it).

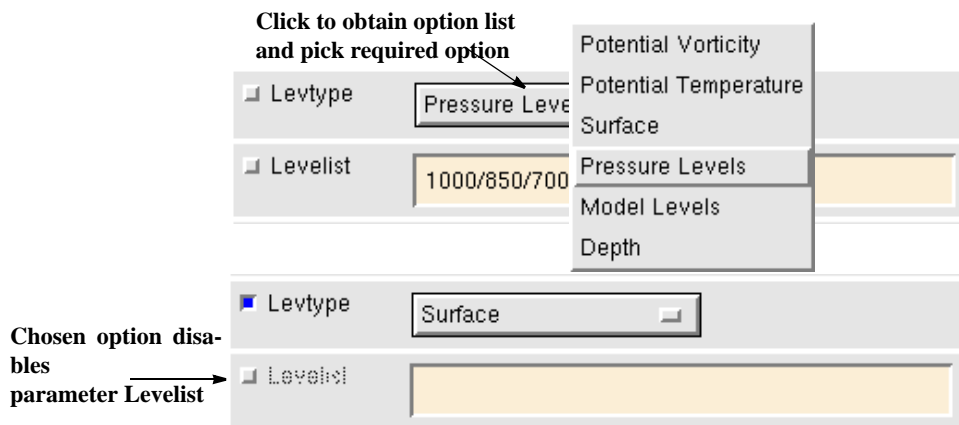


Figure I-19 : Option button on a MARS Retrieval editor window. Click the button to bring up the menu options, select the desired one to make it active. In this case there is active dependency between parameters. Setting Levtype to Surface disables Levelist

When you choose one of the options from a menu, some parameters may become unavailable and others may become available - see Figure I-19 for an example. It may also happen that another parameter value may be changed automatically from its default value.

Icon fields

Some icons require other icons as input, e.g. the Map View icon requires a Coastline icon as input for its *Coastlines* parameter. Icons which require other icons as input have a so called **icon field** where the input icons are placed. Icon fields can accept a number of icons (always of the same type), or accept only a single icon. Most icon fields have a special help feature associated known as the **icon input drawer** - this is a drop down icon drawer, a storage location where users can store frequently used icons of the type required for input - see "Icon input drawers" on page I- 45 for details.

For a detailed account of using icons as input to other icons and functionality of the icon input drawer please see "Embedded Icons" on page I- 44.

To place icons in an icon field either (see Figure I-20) :

- Drop a pre-existing icon from a desktop inside the icon field
- or
- Click-left the trigger for the icon input drawer. Drag one of the icons present in the drawer into the icon field

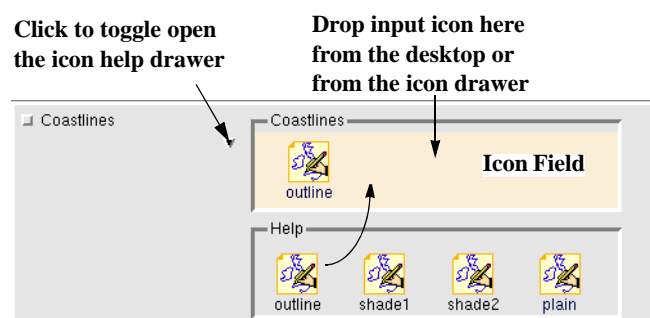


Figure I-20 : Example of an icon field in a Map View editor window, which only accepts Coastline icons. Its associated icon help drawer is open showing a small library of Coastline icons. The icons in the icon field and in the icon drawer may be removed or edited via a two option icon menu.

Colour selection list

Colour selection lists are pick lists of all MAGICS pre-defined colours (stored by their MAGICS names). The colour selection list is accessed via a trigger button which opens and closes the selection list. Users simply click on a colour name to select it.

The colour parameter specified by the colour selection list is either exclusive (e.g. ContourLineColour) or multiple (ContourShadeColourList). In the latter case, newly selected colours are added to the end of the colour list. In the former they replace the current colour.

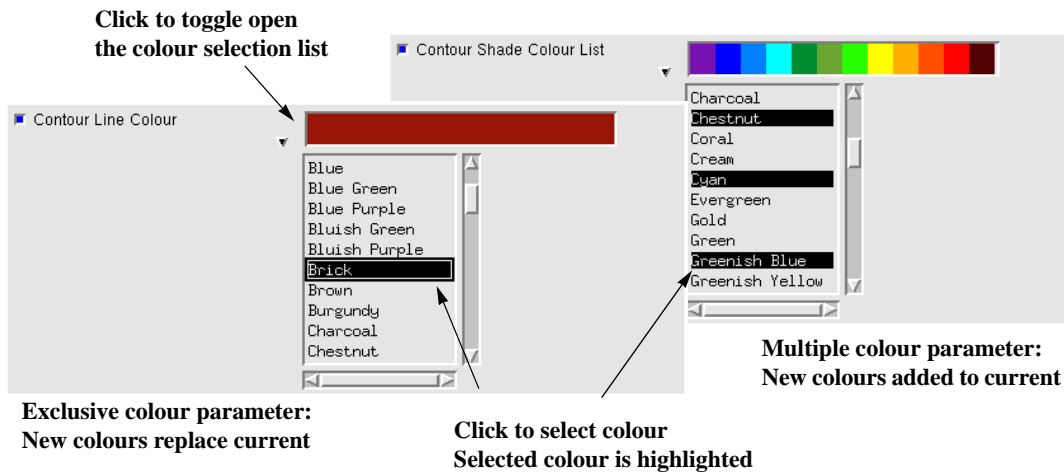


Figure I-21 : Example of colour selection lists in a Contour icon. Selection lists are accessed via a trigger button which opens/closes the colour list. Colour selection can be exclusive or multiple.

Sliders

A slider is an input tool that allows you to specify a numerical value from a (visible) range of possible values. Use the mouse buttons to move the slider as shown in Figure I-22. You may also use the arrow keys to move the slider in unit increments. This may vary slightly depending on how Motif is configured on your machine.

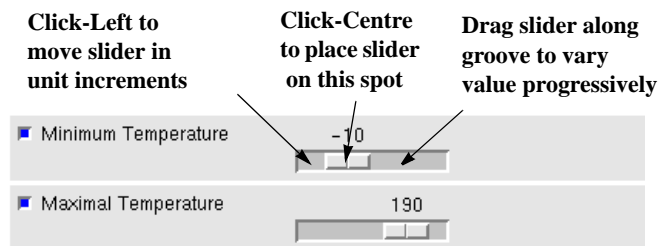


Figure I-22 : Example of sliders from a Tephigram View editor window. The necessary mouse operations are indicated. You can also use the arrow keys.

Editor Window Help Tools

Some parameters have help tools that help you provide the correct input

I/O help tool

The I/O help tool helps specify input and output file names; it launches a directory and file browser. Figure I-23 shows an example where the necessary steps for its operation are outlined.

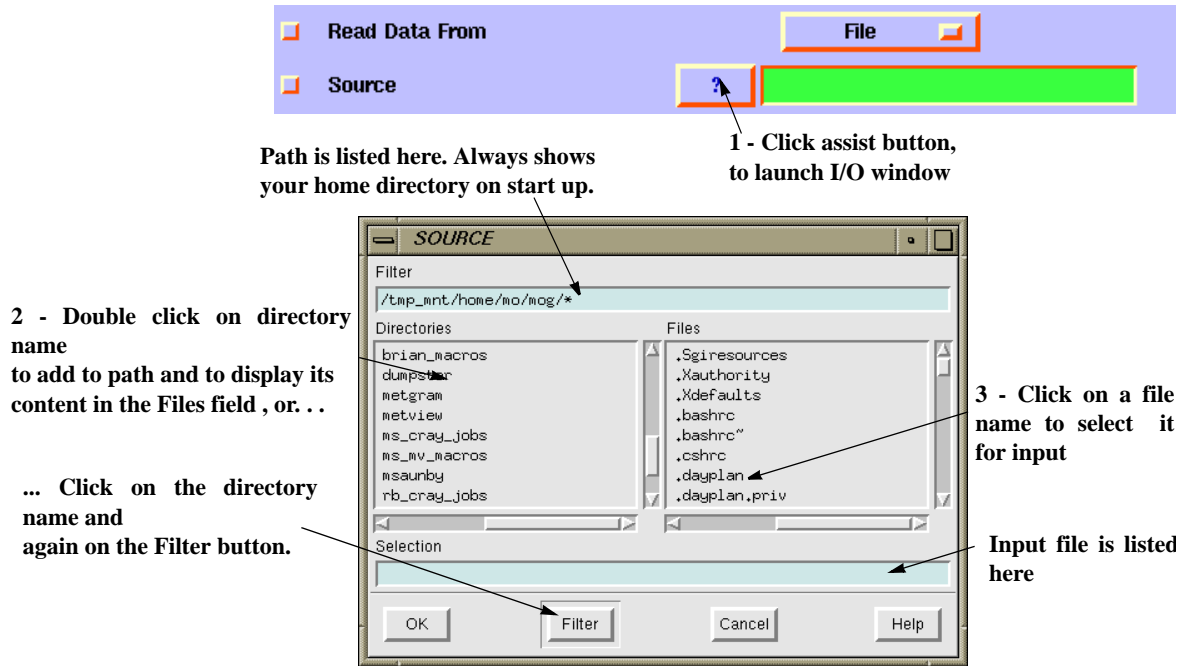


Figure I-23 : The browser launched by the I/O help button in input file mode. In output file mode the browser is similar but the user must type a file name in Selection field.

Geography help tool

Geography help tools ease the specification of geographical elements such as projection type, area limits, line and point coordinates. Whenever an icon input parameter requires area, line or point coordinates, its input element (a text field always) has a help button associated, of which there are three different types - for area, line and point coordinates.

Help buttons for Area, Line and Point :
Click to launch geography help tool

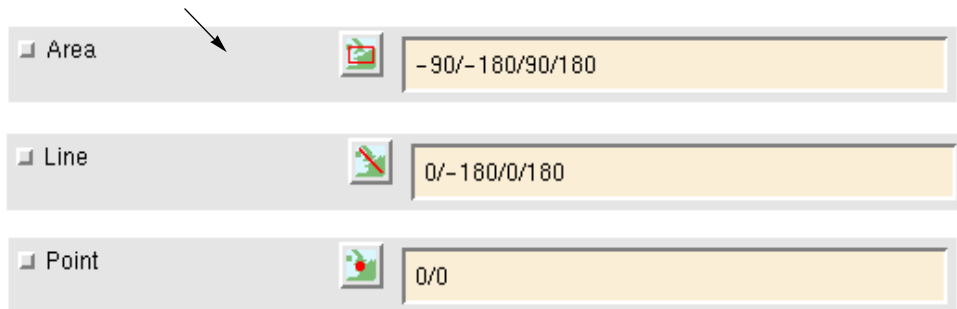


Figure I-24 : The three geography help buttons for area, line and point coordinate selection. Those pictured belong to the icons Map View, Cross-Section View and Vertical Profile View.

A *Click-left* on a geography help button launches an interactive help map window which is similar but with slight operative differences between those for area, line and point selection.

This help map window is known as a **geography tool** and it offers the required functionality for users to customise the map, change projection, to zoom in and out, enter a vertical longitude (i.e. the longitude at the center of the window) and select the geographical element you require - area, line or point. The values and settings derived from this interactive work are passed back to the icon editor window. Full details about working with the geographical help tools are covered in "The Geography Tool" on page I- 91 and "Working with the Geography Tool" on page I- 94.

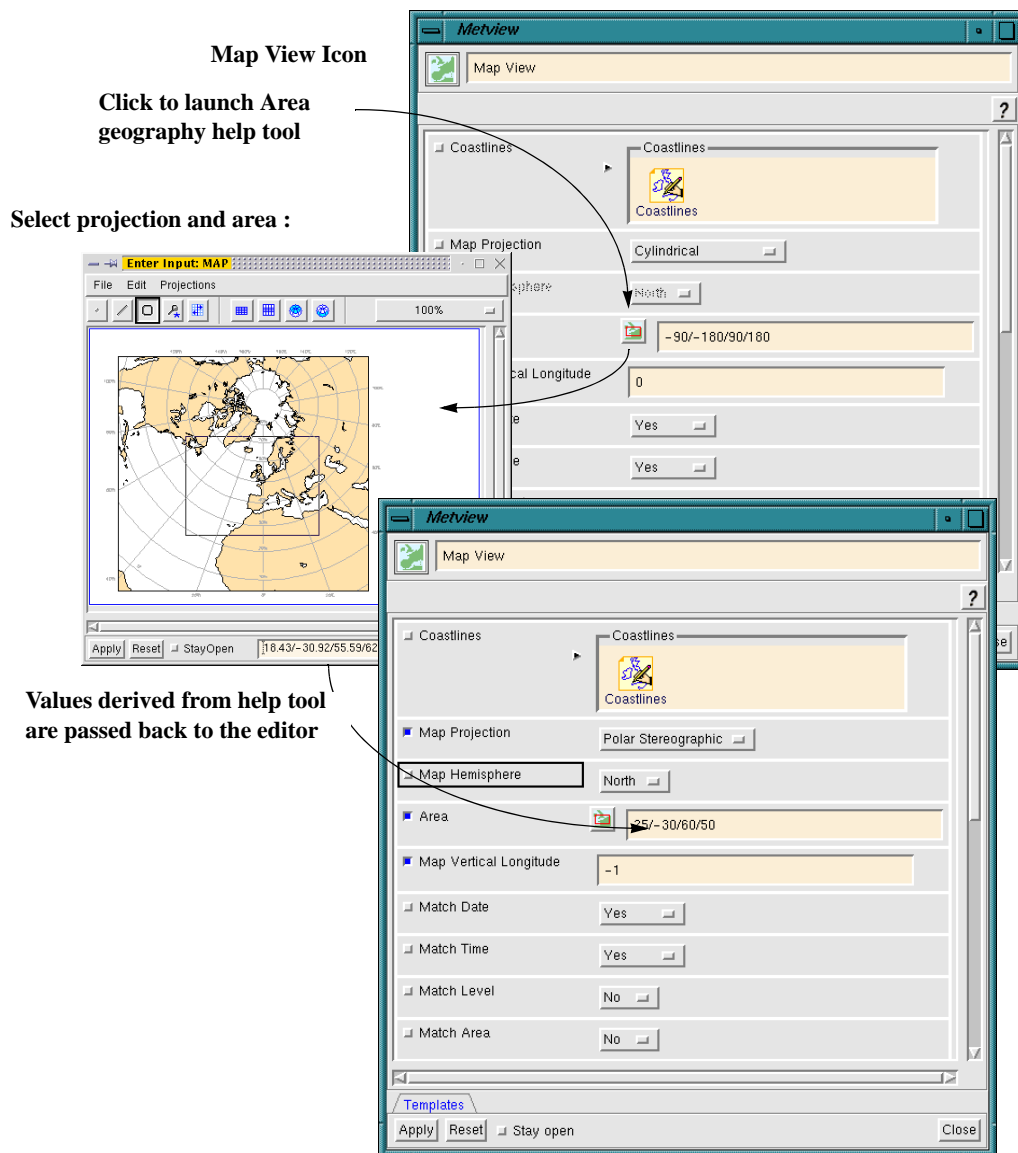


Figure I-25 : Example of area specification in the Map View icon through the geography help tool. The help tools allows you to select a projection, a vertical longitude and a zoom sub-window. The specified sub-area and projection details are passed back to the icon editor.

Station help tool

The Station help button is present only in the Stations icon editor and launches the Metview Stations Database search tool. This can also be accessed via the **Stations** option in the **Tools** menu on the desktop bar. This tool allows you to search the station database using station names, WMO identifiers or location as search keys. For a description, see "Stations" in "The Metview Tools" on page I-133.

Embedded Icons

Some Metview icons require other icons as input, e.g. the Map View icon requires a Coastline icon as input for its *Coastlines* parameter. The input icon is known as the **embedded icon** while the icon which contains it is referred to as the **container icon**.

This principle of icon embedding underpins a lot of the Metview data filtering and computation work : e.g. you could filter some data out of a large file by embedding the datafile icon within a filter icon; the filter icon can then be embedded in an application icon to which it provides input data for some computation. This application icon can still be embedded further as input to other computation icons.

Editor window icon field

Icon fields are storage locations within a container icon's editor window where users drop the required input icons (see Figure I-20 in "Icon fields" on page I- 40). These embedded icons can be supplied from :

- dragged from the icon input drawer (see "Icon input drawers" below)
- dragged from a user desktop
- created *in situ* with the New Icon option of the icon field menu (click-right inside the icon field); this launches the New Icon window from where you must select the icon of the exact type required

Once an icon is in the icon field users can *Edit* or *Remove* them - simply click-right on the icon and choose the required option (see Figure I-26).

What the user sees as an icon field is in reality a hidden folder coupled to the container icon : all Metview icons have a hidden associated directory with the same name but terminated with the character #. Metview makes invisible any folder icon whose name ends with the character #.

Figure I-26 shows an example : the Map View icon NAtlantic is coupled to a hidden directory NAtlantic#, which contains a subdirectory Coastlines, inside which you can find the outline icon; the hidden directory can be opened directly from the icon field using the Open Hidden Folder option from the click-right menu.

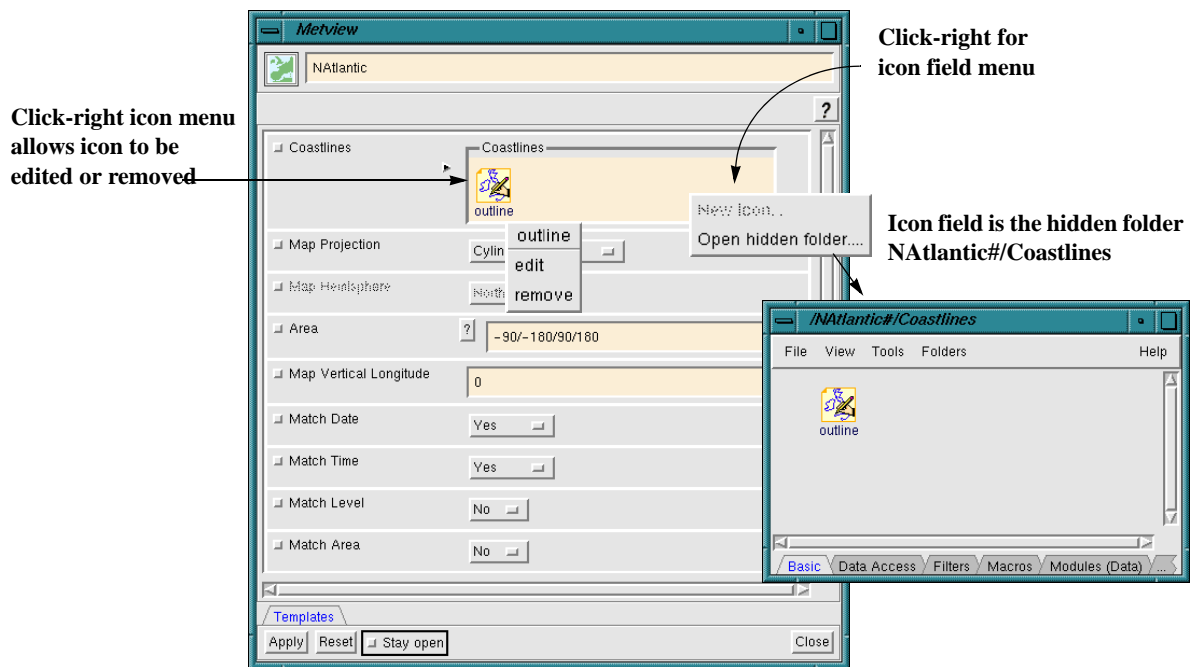


Figure I-26 : Embedded icon in icon editor. Use the icon menu to edit or remove the embedded icon. Also shown is the hidden folder that constitutes the icon field - see text for details.

Icon input drawers

Icon input drawers are storage spaces embedded within icon editors and associated with icon fields; they store (only) pre-defined icons of the required type to be supplied as input (see Figure I-20).

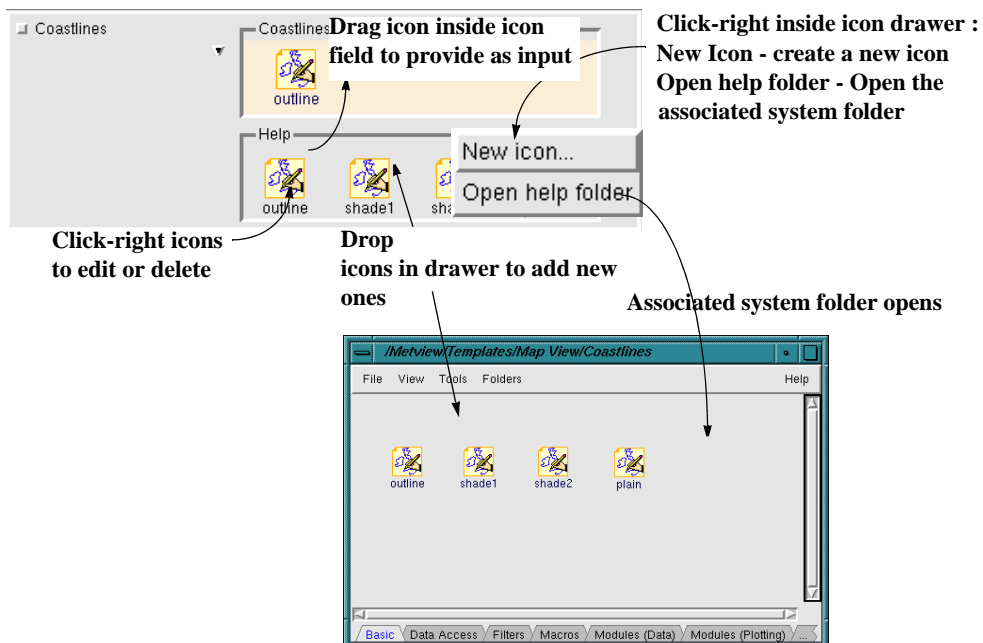


Figure I-27 : The icon input drawer at work. See text for details

The objective is for users to easily build small libraries of icons available for input to a frequently used input parameter - e.g. you may keep most of your Coastline icons inside the icon input drawer associated with the *Coastline* input parameter in the Map View icon, rather than have them clutter your desktop. The icon input drawer is the natural storage location for icons which are (almost) exclusively used as input to other icons.

What you see as an icon input drawer within an icon editor window, is in reality a system directory storing icon templates - \$HOME/metview/Metview/Templates/MapView/Coastlines for the example shown in Figure I-27. Whatever icons are stored in here will appear inside the icon input drawer (provided they are of the recognised type).

The icon input drawer functionality allows you to add new icons to it and modify and remove existing icons. Management of the drawer and its contents is carried out via an icon menu and a drawer menu. As shown in Figure I-27, you can :

- **Add icons** - drag an icon into the drawer (from a desktop or any other drawer) or create a new one in place with the New Icon option from the click-right drawer menu
- **Edit/Rename icons** - to modify an icon directly in the drawer, click-right on the icon and choose Edit from the icon menu. You can rename the icon without editing in the usual way, by clicking the icon name string and amending the name at will
- **Remove icons** - to remove an icon directly in the drawer, click-right on the icon and choose Delete from the icon menu
- **Open the system folder** - choose Open Help Folder from the drawer menu to open the system folder that makes up the icon input drawer

Embedding an icon from the icon input drawer

One way to provide an embedded icon is to drop one of the templates from the icon input drawer into the icon field as shown in Figure I-27. As a result (Figure I-28) the user sees the icon outline residing in the icon field of parameter Coastlines of the Map View icon NAtlantic.

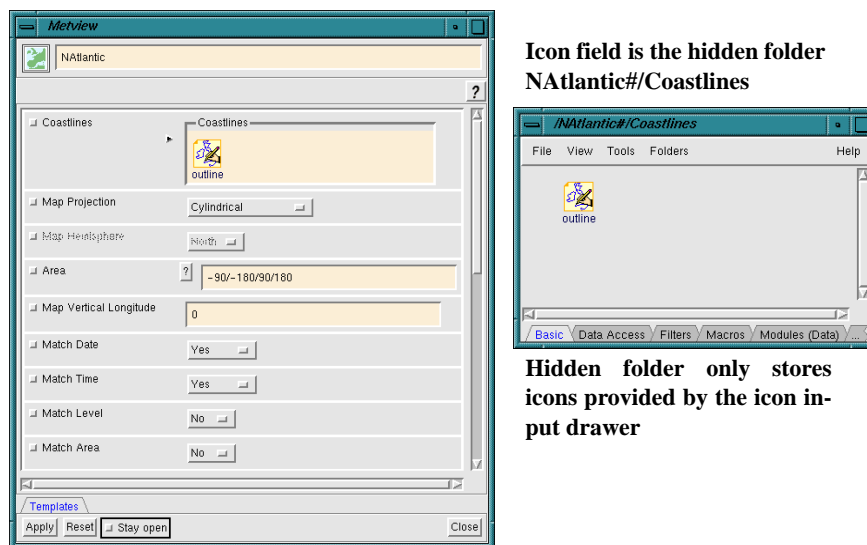


Figure I-28 : Embedded icon in icon editor. The embedded icon originated from the icon help drawer. Use the icon menu to edit or remove the embedded icon. Also shown is the hidden folder that constitutes the icon field - see text for details.

In real terms, what happens is that the icon outline was copied from the system directory \$HOME/metview/Metview/Templates/MapView/Coastlines (the icon input drawer) to the hidden folder ./NAtlantic# /Coastlines (the icon field).

When you move or copy the container icon into another folder, the hidden folder and contents is also moved or copied so that on editing the moved icon you will see exactly the same icon embedded as you had in the original. When you delete the container icon within Metview, the hidden folder and all its contents is removed as well.

Embedding an icon from a user desktop

You can also provide as an input icon any one of your icons residing in a desktop. In this case, the icon you see in the icon field is the *same* as the one on the desktop and nothing is stored in the hidden folder (see above) associated with the container icon. Hence :

- if you edit the icon on the desktop you also change it in the icon field
- if you edit the embedded icon you also change it in the desktop
- if you delete the icon from the desktop, you also delete it from the icon field(s) where it is embedded
- if you delete the icon from the icon field, its desktop instance is left untouched
- If you move or copy the container icon, the link between the icon field and the icon on the desktop is broken and the desktop icon is no longer part of the input of the container icon.

Template Icons

Overview of template icons

When creating an icon you have to modify a number of its input parameters, which can be a long and tedious task, depending on the exact nature of your requirements. A convenient solution to this problem is provided by **Template icons** - they are icons pre-loaded with suitable parameters providing a set of input values that corresponds closely to your requirements. Examples include specific types of MARS Retrievals (e.g. for ensemble members, satellite data, observations, or some "exotic" data), or visual definitions (contouring, arrows or symbols for particular types of data or plots).

The storage location for template icons is the **template drawer** which is an integral part of the icon editor window. Usage of template icons follows the steps :

- Edit the icon you need to modify (just created or previously available)
- On the editor window, open the Template drawer
- Select the template icon you require and import its settings through a merge or a replace operation (see "Using template icons" on page I- 49)
- Introduce any further changes if required
- Save the icon and exit the editor (via Apply)

The result is an icon which contains all or part of the settings of the template icon. You may note that template icons and default icons have some overlap in their roles, in that having a set of icons stored as defaults or as templates may provide the same degree of convenience while fulfilling pretty much the same purpose. In the above steps, you could have a desktop drawer containing the same icons that reside in the template drawer of the icon editor; you would directly create a new icon from the desktop drawer bypassing the template drawer - users should opt for whichever approach they may feel more comfortable with; backwards compatibility with previous versions of Metview also influenced this multiple choice of interface.

Metview provides a number of template icons from the outset, known as **System Templates**, but users can add their own to suit their purposes, thus creating **User Templates**. Template icons can be managed with no restrictions - you can add new ones and modify or remove current templates whether user defined or system provided.

The template icon drawer

Template icons are stored in the template icon drawer which is an integral part of every icon editor window - template icons are used within the icon editing process. The template icons present in this drawer aim to provide sets of input parameters to an icon which is being edited. This is reflected in the different ways you can use the template icons - they can *merge with* or *replace* the icon being edited - see "Using template icons" on page I- 49.

The template icon drawer can store only icons of the same type as the icon editor they are associated with. Typically users store here frequently used or hard to remember definitions, which may not be convenient to store in a desktop. The drawer opens to reveal a variable number of predefined icons. The first times you use Metview you only have available the templates provided by the system, and these are only provided for a few (more important) icons like MARS Retrieval and Contour. However, you can build your own libraries of templates by adding your own icons to the template drawer (see below).

As with other icon drawers in Metview (desktop drawers and icon input drawers), the template drawers also correspond to a system directory, in this case :

```
$HOME/metview/Metview/Templates/IconName
```

where *IconName* is the name of the icon the editor template drawer is associated with.

Management of the template drawer and its contents is carried out via an icon menu and a drawer menu. The following functionality is available when you edit a given icon and then open its template icon drawer opened. You can :

- **Add new icons** - drag and drop an icon into the drawer (from a desktop or any other drawer). The added icons must be of the same type as the editor. You can also turn the icon being edited into a template - click-right and choose the option Make a template from current definition from the drawer menu; this allows you to build your own library of template icons
- **Edit/Rename icons** - to modify an icon directly in the drawer, click-right on the icon and choose Edit from the menu. You can rename the icon without editing in the usual way, by clicking the icon name string and amending the name at will
- **Remove icons** - click-right on a template icon and select Remove from the icon menu; system template icons are read-only and can't be removed
- **Open the template folder** - choose Open Template Folder from the drawer menu

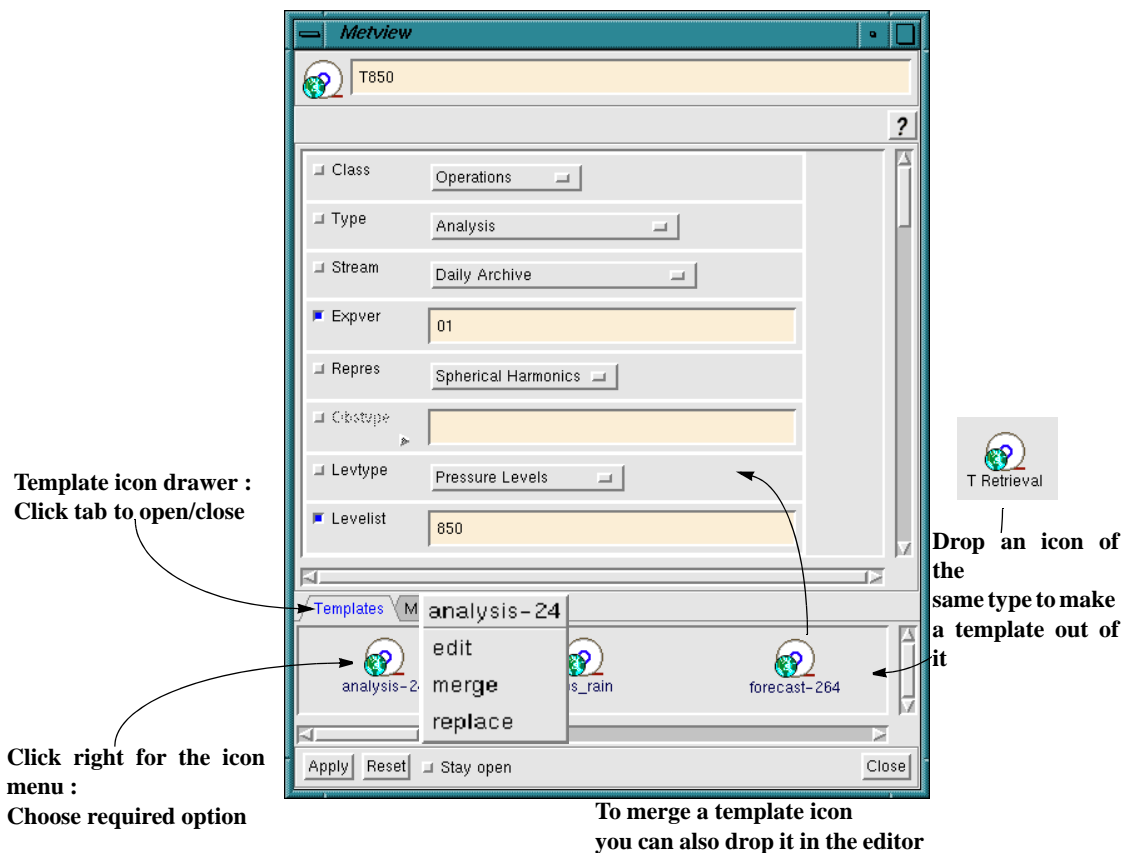


Figure I-29 : Importing settings from template icons. You can either merge or import the settings of a template icon from its icon menu.

Using template icons

To use the icon editor templates click open the Templates drawer, situated at the bottom of any icon editor window.

There are two basic ways to incorporate settings from a template icon into the icon you are editing, as shown in Figure I-29 - *Merge* and *Replace*.

- **Merge** - when you merge the settings of your icon with those of the template icon *only the non default parameters of the template icon overwrite the corresponding parameters in the destination icon.*
- **Replace** - when you replace the settings of your icon with those of the template icon *all the parameters of the template icon overwrite the corresponding parameters in the destination icon*, irrespective of whether the parameters are default or non-default.

Both options are available from the template icon menu :

- Click the Templates drawer tab to open it, identify the required template icon and click-right on the icon to bring up its icon menu
- To *merge* the settings, select Merge from the template icon menu, or simply drag the template icon into the input area
- To replace the settings, select Replace from the template icon menu
- Once the settings have been merged or replaced, save and exit the icon as usual

Note that merging allows inconsistent specifications to be made - e.g. if a contour icon has parameter Contour set to Off (non-default) and you merge it with a template which has Contour set to On (default), Contour Line Thickness and Contour Line Colour set to some non-default settings (say 3 and Black) the resulting icon settings will include these last two but with Contour still set to Off. This is inconsistent in that you specify contour line characteristics while at the same time turning the contour lines off.

Merging with desktop icons

You can merge the settings of the icon being edited with those of any desktop icon - simply drag the desktop icon and drop it inside the opened editor window. Metview checks the type consistency of the icons involved and the procedure is only allowed for icons of the same type.

The settings of the dropped icon are imported into the destination icon by merging only, i.e. *only the non default parameters of the desktop icon overwrite the corresponding parameters in the icon being edited.*

Editing Icons in ASCII Mode

As detailed in "Icons Revealed" on page I- 32, what you see as an icon in the Metview environment, is actually composed of two files - the **dot file** and the **icon file**.

The icon file is a native ASCII file that stores the icon parameter list (while the dot file stores icon type and status information). The Metview icon editors allow you to have access to the icon file itself, i.e. to edit directly its ASCII contents instead of the usual way via the editor input elements.

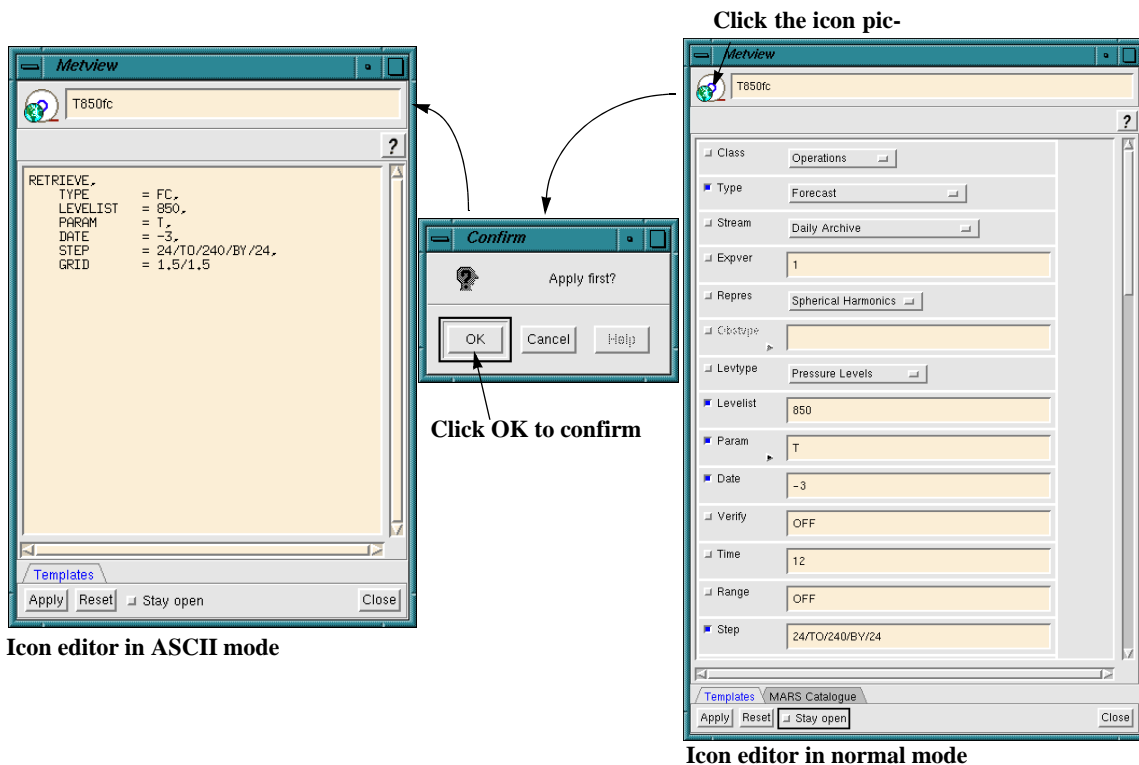


Figure I-30 : Using the icon editor in ASCII mode - click on the icon picture to launch the editor in ASCII mode. The ASCII contents will be familiar to users of MARS and MAGICS

To access the ASCII icon content :

- Click once on the icon picture at the top left of the icon editor window
- Click OK when prompted to whether you want to apply first (i.e. save the changes made so far)

The editor window input area turns into a plain text editor. This is known as **editing in ASCII mode**. Note that icons such as Notes, Shell, Formula and Macro are already edited in ASCII mode by definition. For icons other than these, editing in ASCII mode is not recommended - the icon editors proper provide a far more error-proof interface.

Figure I-30 shows the process applied to the icon of the example in "Icons Revealed" on page I- 32. Compare the icon file contents listing presented there with the contents of the editor window in ASCII mode shown in the figure - this shows that when you edit in ASCII mode you edit the contents of the icon file itself.

Given the limited capacities of text editing in Metview, once you have the icon being edited in text mode, you can launch an alternative text editor - see "Using alternative editors" on page I- 52.

Working with Text Icon Editors

The simplest editors you have in Metview are the plain text editors of the icons Notes, Shell, Formula and Macro. The text editor in Metview is limited to very simple operations (listed below) but

this is sufficient for most needs involving the icons Notes, Shell and Formula, which use only small amounts of text. The Macro icon is the only icon on which users may have to spend reasonable editing efforts and its editor provides users with specific code editing tools such as syntax checking and beautifying.

However, should you need better text editing facilities for your macro programs, you can use any text processor you want instead of the Metview macro editor. There was little point in reinventing the text editor wheel, so the facility to use an external one was implemented instead.

Handling text

Input - Click inside the editor and type away to input text from the cursor position

Selecting text - Both *Shift-arrow* or *Drag-left* from one point to another will select all text in between. Double *click-left* will select a word, treble *click-left* will select the entire line of text and four *click-left* (in quick succession) will select the whole text.

Copy - Any text which is selected is copied to the buffer.

Cut - There is no Cut action, you have to Copy, Paste, Delete.

Paste - Move the mouse pointer where you want the text to be placed and *click-centre* to paste. Note : *Do not click* any other mouse button or press the arrow keys (e.g. to move the text cursor) before pasting, otherwise the buffer is emptied. However, when pasting to another editor, you should first click on the destination window to define the insertion point of the pasted text.

Replace - Anything you type while text is selected, will replace the selected text.

Delete - Press the Delete or Backspace keys to delete the selected text

Using alternative editors

If you prefer, you can use an alternative (external) text editor (such as vi, emacs, jot, xedit, ...) rather than the Metview text editor window. You can do this from the Metview text icon editor of Notes, Shell, Formula, Macro, or any icon being edited in ASCII mode (see "Editing Icons in ASCII Mode" on page I- 50).

Before you start using an alternative editor, you have to set up the one you want to use - from the UNIX prompt, check if the \$EDITOR environment variable is set :

```
% echo $EDITOR
```

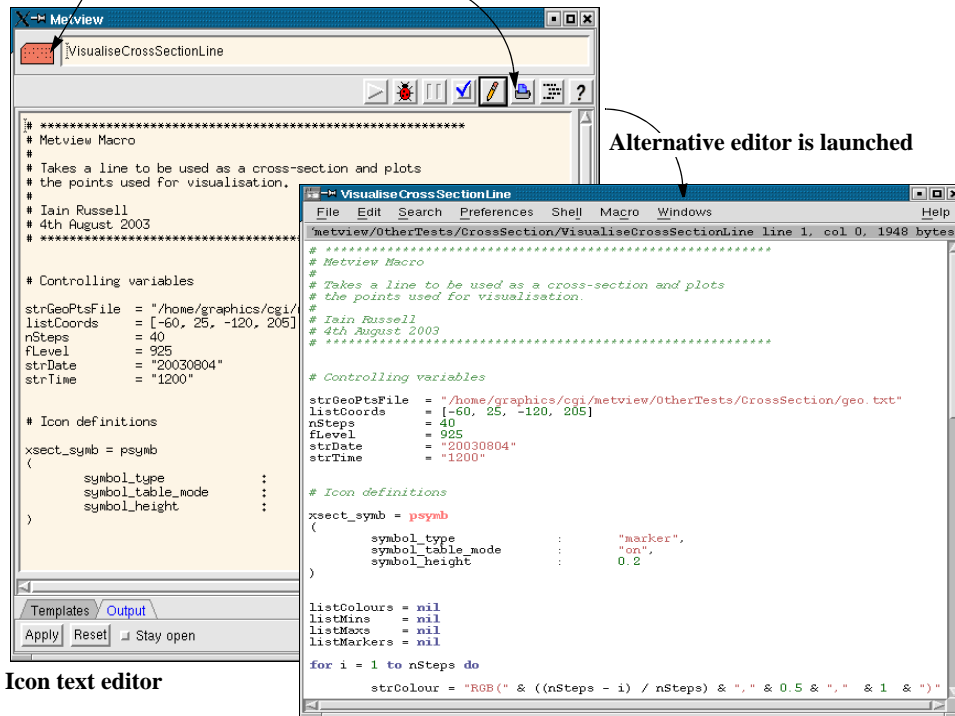
if it is, this prints the name of the external editor (vi, emacs, xedit, jot) currently set up. If the editor is the one you want, leave it as it is, otherwise or if not set, do one of the following :

```
% setenv EDITOR editor_name (vi,emacs,...) C-shell
% EDITOR = editor_name (vi,emacs,...) Bourne-shell
% export EDITOR=editor_name (vi,emacs,...) Korn-shell
```

If you wish to use a particular alternative editor within Metview but leave your default editor unchanged, you may instead set the environment variable METVIEW_MACRO_EDITOR, following

the same procedure as shown above. If METVIEW_MACRO_EDITOR has not been set, then Metview will use EDITOR.

Click on icon picture or on the pencil button



Icon text editor

Figure I-31 : Using an alternative editor, in this case NEdit. You can use these for macro programs (mostly) but also for other icons, provided they are being edited in ASCII mode.

This must be done *before* you start Metview, otherwise exit, set the editor and restart again. Once set up, do as follows to use your alternative editor :

- Edit the icon in the normal way to launch the ordinary Metview text editor
- Launch the alternative editor with a click-left on the icon picture (top left of the icon editor window), or on the pencil work button in the Macro editor window

The alternative editor window comes up, displaying the text contents of the icon; edit at will and save/exit in the normal way. Note that you can't go from the alternative editor back to the standard editor, you have to close the alternative editor and *Edit* the icon again.

Metview provides a way to enable full macro syntax highlighting using the text editor 'NEdit' as the alternative editor - see "NEDIT Syntax Highlight 2.0" on page III- 83 for details.

Drops into text icon editors

A feature of Metview text editors is that they accept drops of other icons. This is of interest mostly for the case of the Macro editor, in that it can greatly speed up the conversion of an interactive task into a Macro program. Indeed, this is a recommended way of starting Macro programming - see the Macro language manual for details.

The general behaviour of Metview text icon editors (Notes, Shell, Formula and Macro) in response to icon drops is to convert the elements of the dropped icon into text; the exact details of the resulting text depend on which text icon is involved :

- Icon dropped in a Notes or Shell editor - the exact text contents of the dropped icon are inserted at the cursor position (see "Icons Revealed" on page I- 32 for details on text contents of icons).
- Icon dropped in a Macro editor - again the text contents of the dropped icon are inserted at the cursor position, but obeying the Macro language syntax
- Icon dropped in a Formula editor - only the name of the dropped icon is inserted at the cursor position

As an example, consider the example icon given in "Icons Revealed" on page I- 32, a MARS Retrieval icon of T850 forecasts, named T850fc. If this icon is dropped in a Formula editor, the result is simply (details may vary depending on whether the dropped icon is in the same folder as the Formula icon) :

T850fc

If it is dropped in a Notes or Shell editor, the result is identical to its text contents, i.e. :

```
RETRIEVE,
TYPE    = FC,
LEVELIST = 850,
PARAM   = T,
DATE    = -3,
STEP    = 24/TO/240/BY/24,
GRID    = 1.5/1.5
```

If it is dropped in a Macro editor, the result is the same but the syntax is that of the macro language :

```
t850fc = retrieve(
           type      : "fc",
           levelist  : 850,
           param     : "t",
           date      : -3,
           step      : [24,"to",240,"by",24],
           grid      : [1.5,1.5]
        )
```

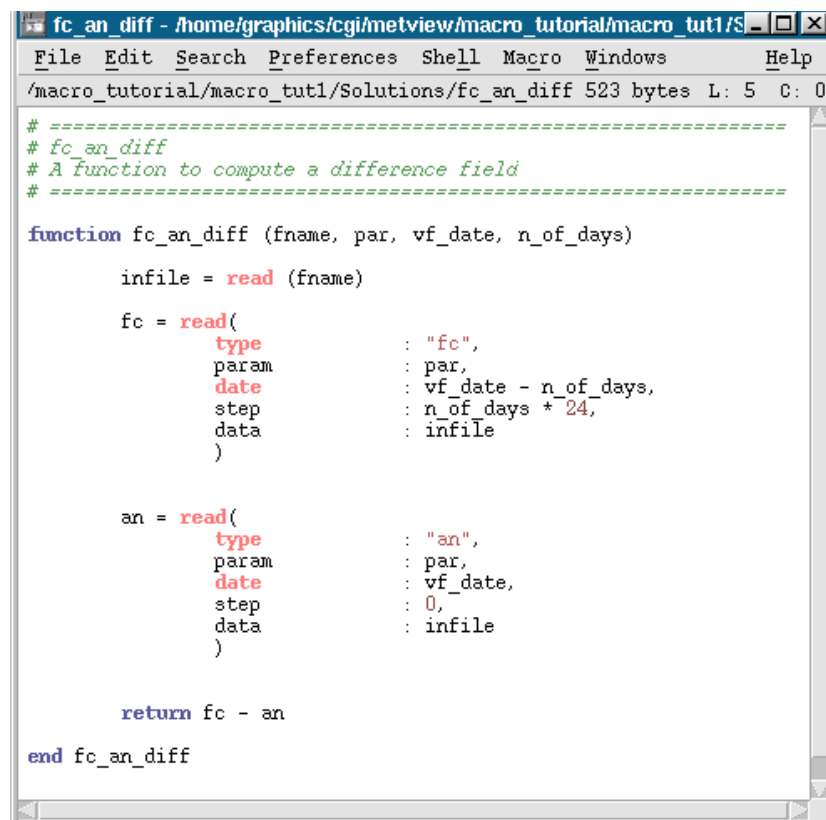
Using Customised NEdit as the Alternative Editor

It is possible to set up the text editor NEdit so that it can be used as an advanced macro editor - see "NEDIT Syntax Highlight 2.0" on page III- 83 for details of how to set it up. Once this has been done, it can be launched as the alternative text editor from within the built-in macro editor toolbar - see "Using alternative editors" above. The **Macros** menu contains a complete list of the available Metview-specific functions, detailed below after Syntax Highlighting.

Syntax Highlighting

For clarity and ease of editing, the following components of the macro language are displayed in separate colours and/or styles:

- keywords
- comments
- built-in functions
- strings
- numbers
- ECMWF macro library functions (mvl_*)



```

# =====
# fc_an_diff
# A function to compute a difference field
# =====

function fc_an_diff (fname, par, vf_date, n_of_days)

    infile = read (fname)

    fc = read(
        type      : "fc",
        param     : par,
        date      : vf_date - n_of_days,
        step      : n_of_days * 24,
        data      : infile
    )

    an = read(
        type      : "an",
        param     : par,
        date      : vf_date,
        step      : 0,
        data      : infile
    )

    return fc - an
end fc_an_diff

```

Figure I-32 : Syntax highlighting with NEdit.

Calltips

A calltip is a text box that pops up, giving useful information about a function. To invoke a calltip, first indicate which function you would like help with. This can be done by either selecting the function name or by placing the caret within or just after the function name. Now, press F1; if a calltip exists for the given function, it will appear. Note that this functionality is only available with NEdit version 5.4 and above.

```

Step4 (modified) - /home/ectrain/trx/metview/macro_tutorial/macro_tut1/
File Edit Search Preferences Shell Macro
/home/ectrain/trx/metview/macro_tutorial/m
# set the required variables

home          = getenv ( "HOME" )
pa1any getenv ( string )
fi1string getenv ( string, number )
Returns the value of an environment variable given its name as the argument.
If a second argument (number) is given and the number is zero,
pa1 the function returns a string, even if the environment variable content
vf looks like a date or a number.
n_of_days     = 5
    
```

Figure I-33 : Using calltips with NEdit - select a function then press F1.

Listing of built-in functions

Metview’s macro language contains many built-in functions. You can access a list, sorted by category, of all the available functions by pressing F2. From this list, either double-click the function you wish to insert, or click once on it followed by the **Insert** button.

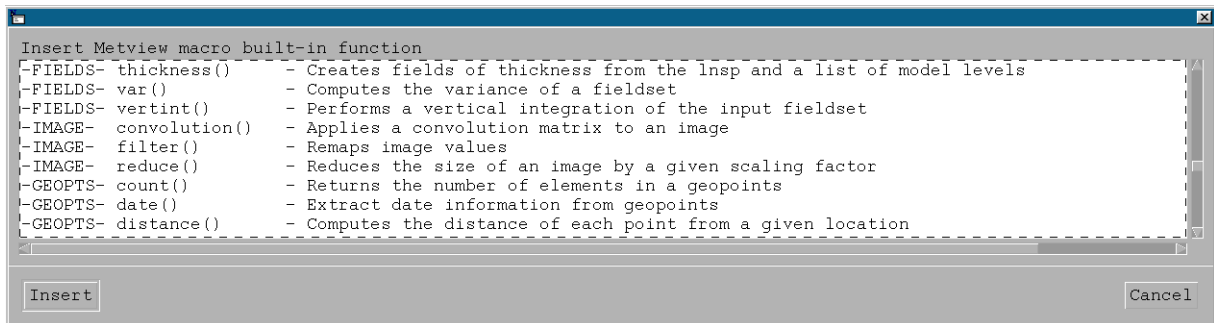


Figure I-34 : Invoking the list of built-in functions - press F2.

Listing of available library macros

If there are appropriately-formatted macros present in the macro search path (see "Using function look-up" on page -48), these can be accessed in a similar way to the built-in functions. Press F3 to bring up a list of all such available macros. Consider the following example of a macro function (see Figure I-35). It contains a comment header section consisting of various fields. These fields are then used for two things:

- creating an entry in the list of available macros (see Figure I-36)
- creating a calltip for the macro (see Figure I-37)

```

# mvl_regular_layout - /home/graphics/cgi/metview/Metview/Macros/
File Edit Search Preferences Shell Macro Windows
/home/graphics/cgi/metview/Metview/Macros/mvl_regular_layout 2369 bytes

# *****
# Function      : mvl_regular_layout
# Syntax       : list mvl_regular_layout (definition, number, number,
#              :              number, number)
#
# Author (date) : Anonymous (--/--/2003)
#
# Category     : LAYOUT
#
# OneLineDesc  : Generates a regular grid of frames/subframes
#
# Description  : Creates a list of frames, arranged in a regular grid.
#              : Each frame contains a set of 1 or more subframes, each
#              : arranged in a regular mxn grid. The output is suitable for
#              : input into the function plot_superpage().
#
# Parameters   : the_view      - a view definition to be used with each frame
#              : frame_cols    - the number of frames along the width
#              : frame_rows    - the number of frames along the height
#              : subframe_cols - the number of subframes along the width
#              :                of each frame.
#              : subframe_rows - the number of subframes along the height
#              :                of each frame.
#
# Return Value : The list of frames
#
# Dependencies : mvl_mxn_frames
#
# Example Usage :
#               # create a set of frames using the default map view
#               page_list = mvl_regular_layout (mapview(), 2, 1, 1, 3)
#
#               # create a display window using this set of pages
#               dw = plot_superpage( pages : page_list )
#
# *****

function mvl_regular_layout (the_view      : definition,
                             frame_cols   : number, frame_rows   : number,
                             subframe_cols : number, subframe_rows : number)

    delta_x = 100 / frame_cols
    delta_y = 100 / frame_rows

    subframe_list = mvl_mxn_subframes (subframe_cols, subframe_rows)

```

Figure I-35 : An example macro formatted for inclusion in NEdit's macro list.

A colon separates a field's name from its value; the value of a field can take up multiple lines and its spacing will be kept. Tabs will, by default, be translated into groups of 8 spaces. The following fields are necessary in order to generate a list entry for the macro:

- Function - provides the name of the function.
- Category - the list of available macros is sorted by category in order to make searching the list easier. This field can, in theory, be anything; however, it is best to stick with existing categories unless it is necessary to create a new one as too many categories will make the list more difficult to navigate.
- OneLineDesc - this is a one line description of the function, designed to give the user a good idea of what it is used for.

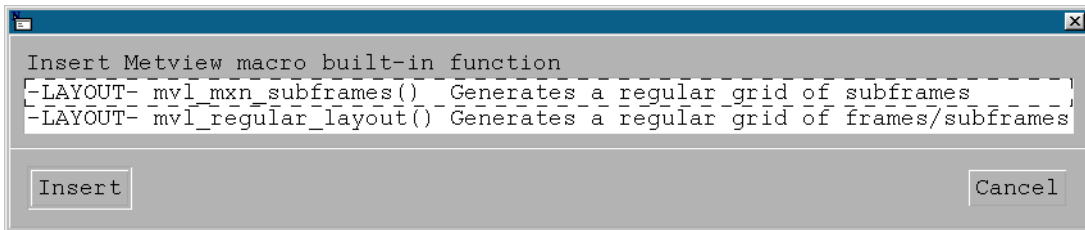


Figure I-36 : An example list of available macros - press F3 to invoke.

In order to generate a calltip for a macro, only the Function field *must* be present. However, the following fields will be used if present:

- Syntax - a description of the input parameters' types
- Description - a full description of the macro; what it does and how it works
- Parameters - a description of each input parameter
- Return value - a description of the data that the macro returns
- Example usage - some example code showing how to use the macro

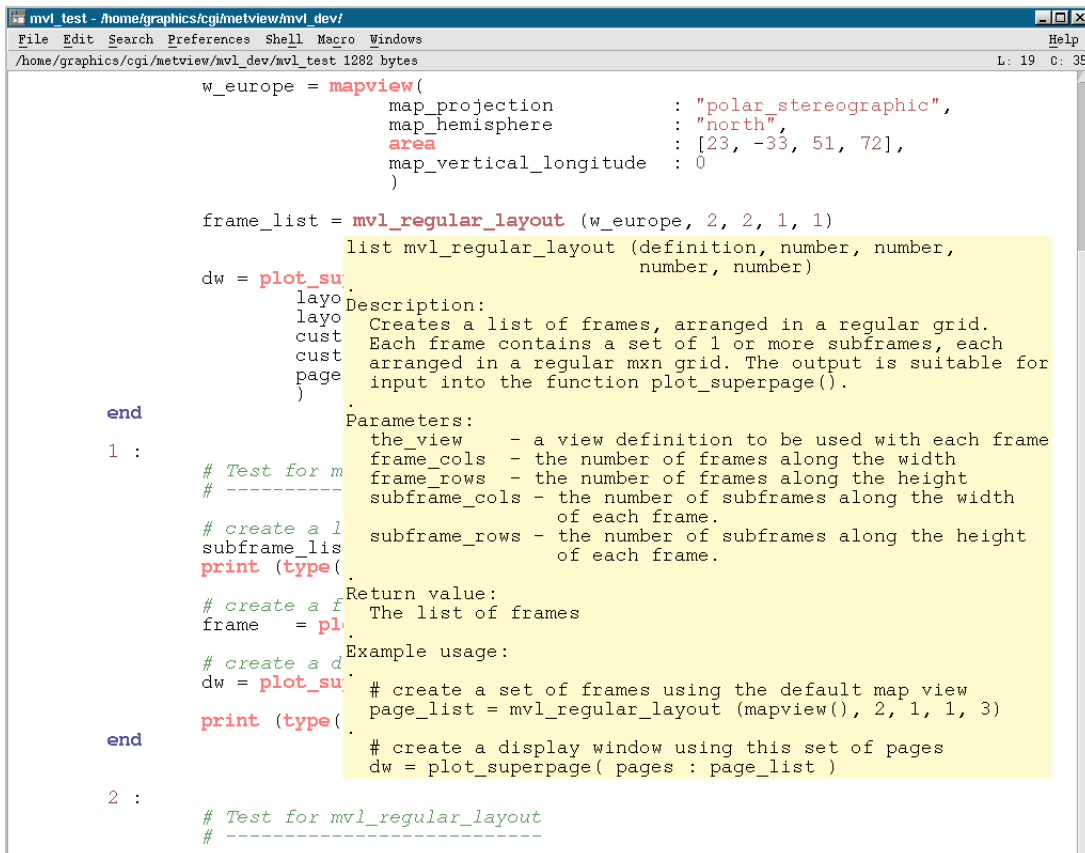


Figure I-37 : Example of a macro calltip generated from header comments in the macro.

Once a macro (or more) in the macro search path have been modified to include this style of header, you must update NEdit's Metview-specific files. See "Updating NEdit's Metview-specific files" on page I- 60 for more details.

Code templates

It is possible to store fragments of code for easy insertion into your macro programs. Pressing F4 will bring up a list of all available code templates; from this list, either double-click the template you wish to insert, or click once on it followed by the **Ok** button.

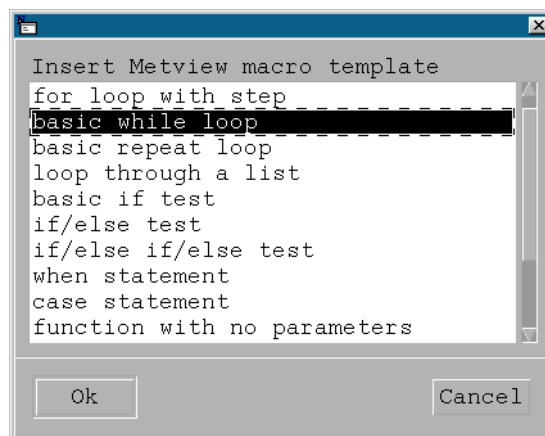


Figure I-38 : A code template list - press F4 to invoke.

There are three sources of templates:

- default, ECMWF-supplied templates (these come installed with Metview)
- local site-supplied templates
- user-supplied templates

The first two are located in the 'etc' directory of your Metview installation and should not be touched by users; however, the default template file may be consulted as a reference. The user templates are in `~/metview_aux_files/macro_templates_user.txt`.

These files share a common format. A template definition starts with a brief description in curly brackets. This is the text that will appear in the template list box. The rest of the text is what will be inserted when the user chooses that template.

```
{for loop with step}
for _ = _ to _ by _ do
  <code>
end for
```

```
{basic while loop}
while _ do
  <code>
end while
```

Once a template file has been modified, you must update NEdit's Metview-specific files. See "Updating NEdit's Metview-specific files" on page I- 60 for more details.

Importing Icons with NEdit

A limited ability to incorporate macro equivalent code from icons dropped in a NEdit window is included. Drag one or more Metview icons into a NEdit window, then press F12. The macro equivalent source code for the icon(s) will appear at the cursor. If this does not work immediately, wait for a second or two and try F12 again - there is a dedicated service module that starts up the first time you drag an icon into a NEdit window.

Commenting blocks of code

In order to quickly comment out a block of macro code, select (highlight) the lines and choose **Comment Selection** from NEdit's **Macro** menu. To uncomment a block, select it and choose **Uncomment Selection**.

Updating NEdit's Metview-specific files

In brief, run **Metview Refresh Lists, Calltips & Templates** from NEdit's **Macro** menu when any of the following have been changed:

- the list of built-in functions
- the calltips for the built-in functions
- any header of a macro in your macro search path
- any of the three code template files

This option is only available when a Metview macro is being edited or if NEdit's 'Language Mode' has been manually set to Metview_Macro. Installation of a new version of Metview will very likely be an occasion where this is necessary. Some user actions may also cause it to be necessary; the reasons should become clear with greater explanation of the macro, below.

The NEdit macro runs a script that checks every file in every directory in your macro search path (environment variable \$METVIEW_MACRO_PATH, only defined when running Metview). For each macro file that contains an appropriate header, a list entry and calltip are generated. Note that the list of available macros is specific to the user directory with which Metview was started (see "Starting Metview" on page I- 3). This is because, by default, starting Metview with a different user directory causes a different set of directories to be included in the macro search path. Thus, if you use Metview with different user directories, you will need to perform this procedure for each one; however, you will not need to do this each time you switch between user directories that you have already set up in this way. At the end of this process, a list of files that could not be documented is shown.

Running this NEdit macro also combines the three template files (see "Code templates" on page I- 59) into the one that is used within the editor. This means that if changes are made to any of the three template files, then this procedure will have to be followed in order to have any effect.

If you switch between different Metview user directories, you may also need to run **Metview Macro Update Calltips Link** from NEdit's **Macro** menu. If you have different calltips for different Metview user directories (eg because of different macros in the macro search path), then NEdit needs to know which one to use; this NEdit macro sets the link to the correct calltips file for the current Metview user directory. Run this macro each time you start Metview with a different user directory. **Metview Macro Update Calltips Link** is called automatically by **Metview Refresh Lists, Calltips & Templates**.

VISUALISATION - AN OVERVIEW

Introduction

This chapter introduces some basic concepts that underpin much of the visualisation work in Metview. Following chapters cover details of the visualisation tool, visualisation layout design and the actual visualisation work.

Users of previous versions of Metview will identify major improvements in the current visualisation functionality relative to that of previous versions, such as :

- Ability to mix different types of plots in any arbitrary layout on the same display (screen/file/paper)
- Effective WYSIWYG capacity between on-screen and hard-copy displays
- Much greater flexibility in the manipulation of visualisation components

The flip side of greater flexibility is that setting up and preparing visualisations becomes more complex. Metview visualisations are executed on a **Display Unit**, which can be :

- An interactive Display Window on screen
- A graphical file on disk (PS/JPEG/PNG)
- A hardcopy output (paper/transparency)

Most detail in this manual covers the Display Window as the display unit. Graphics files / hardcopy output display units can be generated from the on-screen display window or directly via a macro program.

Metview can visualise data in one of the recognised **data formats** :

- WMO GRIB (editions 1 and 2) format for fields (note that GRIB 2 files are internally converted to GRIB 1 before visualisation)
- WMO BUFR format for observations
- The extended WMO GRIB format for satellite images
- ASCII matrices of lat-long gridded data
- ASCII lists of irregularly spaced data points (Geopoints)
- NetCDF for data other than model fields and point data (e.g. cross sections)

Metview allows users to produce the following **types of plots** :

- 2-D geographical plots - fields, observations, station locations, satellite images
- vertical cross-sections along defined spatial limits
- vertical profiles of parameters for points or areas
- time series of parameters for a given point
- tephigrams
- a variety of XY or geographic curves

Approaches to Visualisation Work

Visualisation work in Metview can be approached in two ways depending on the complexity of the required end-product and the nature of the visualisation task. We designate these two approaches to visualisation work by :

- **Direct Visualisation** - for quick single-plot visualisations, e.g. in data checking and exploratory analysis
- **Layout Visualisation** - for visualisations involving more than one plot, usually for more specialised analysis and for preparation of presentation plots. You need to prepare a display layout where you subdivide your display area in a number of *plot frames* in a given regular or arbitrary arrangement

The two types are described in detail in "Direct Visualisation" on page I- 68 and "Layout Visualisation" on page I- 69. Whichever type one considers, visualisation work involves a set of icon types whether directly (when users create the icons to be used) or indirectly (when the system supplies suitable defaults). The icons used are :

- **Display Window icon** - this specifies the layout and characteristics of the display. This icon is explicitly involved only in Layout Visualisation, since a direct visualisation will use the system default layout (landscape A4, with a single map).
- **Data Unit icons** - these provide what it is to be plotted and include both icons of data files (GRIB, BUFR, ASCII Matrix) and all icons that can return data; icons that return data comprise database retrievals (MARS), data filters, applications and suitably coded macro programs.
- **Visual Definition icons** - these provide the plotting characteristics, e.g. contour specifications (line colour, colour shade list, contour interval, ...), observation plotting elements, axis details (colour, tick marks), text (for titles and annotations), etc. Again if not provided, system defaults will apply.
- **View icons** - these specify the type and characteristics of the plot to be produced from the data and are embedded in the Display Window. View icons are only specified explicitly in Layout visualisations, since a direct visualisation will use the system default view applicable to the data unit - you can change this by dropping another view icon in the display window.

Data Unit - Input to Visualisation

The first fundamental element in a Metview visualisation is the **Data Unit**. A data unit is simply a set of data values in some arrangement and format. A data unit can be :

- data files on disk (model fields in a GRIB file, observations in a BUFR file, etc)
- data requests from a database or storage space (e.g. MARS retrievals)
- output from a data filter (subsets of large model field sets, single/double parameter geopoints out of BUFR file of observations)
- output from a computation (data generated by macro icons or application icons)

Since everything in Metview is an icon, each type of data unit is represented by an icon. The icons that act as data units are :

- **Database request and data filtering icons** - MARS Retrieval, GRIB Filter, ObsFilter, Ecfs, Geopoints to GRIB, GRIB to Geopoints
- **Data file icons** - GRIB, BUFR, geopoints, Matrix, ASCII lat-long data grids and NetCDF files
- **Computation icons** - Simple Formula, Formula, Macro (provided it generates data), MacroParameters
- **Application icons** - these include all the icons which carry out a specific computation on some input data to produce a derived data unit; e.g. the Potential Temperature icon which outputs a GRIB of potential temperature, Cross-Section which outputs a NetCDF file of cross-sectional data (point values along a transect for a number of vertical levels)

Visualisation in Metview is simply to derive plots from data unit icons. Users can carry out a visualisation in two ways - **directly**, i.e. straight from the data unit icon by means of the **Visualise** option in its icon menu or through a process of **layout design** which allows users to display several types of plots in any arbitrary arrangement in a display unit - this is detailed in "Direct Visualisation" on page I- 68.

Depending on the data unit, you may be able to produce from it a single well defined plot or a wide variety of plots. E.g. from a NetCDF file of cross section data you can only produce a cross-section plot, but from a GRIB file of multi-level, multi-forecast step model fields, a wide variety of plots can be produced (cross-section, vertical profile, zonal averages, maps in a variety of projections and sub-areas, etc.).

In Metview, users determine which plots are produced from a data unit by means of a user defined display specification known as a **View**, which also provides essential plotting details (grids, coastlines, axis, ...). Views underpin the visualisation work in Metview and you always use them (even if not explicitly). The concept of View is detailed fully in "The View Concept in Visualisation" on page I- 65.

Visual Definitions - Plotting Control

The second fundamental element in a visualisation is the **Visual Definition**. Visual Definitions are sets of plotting instructions and as everything in Metview they are specified by means of icons. Available visual definition icons are pictured below :

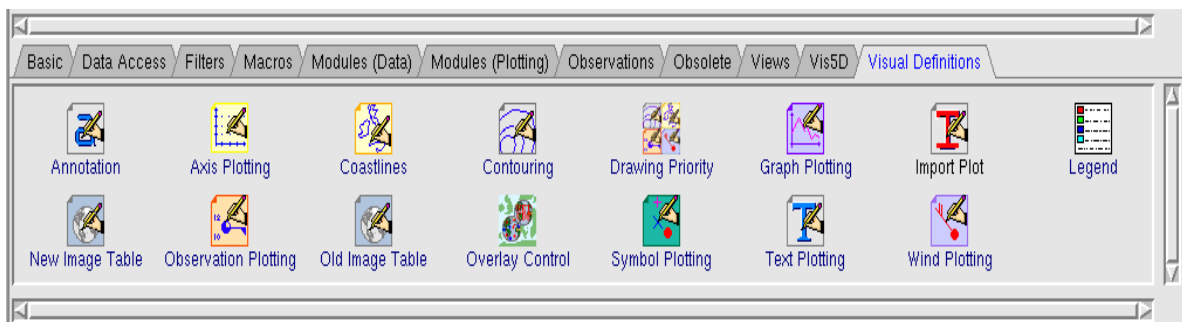


Figure I-39 : Visual Definition icons residing on the Visual Definitions desktop drawer

To produce a plot from data unit icons you need to define and supply visual definition icons. Even if you don't do it explicitly (e.g. when visualising a data unit directly), Metview will always provide a default visual definition icon (which depends on the type of data returned by the data unit icon to visualise).

Visual Definition icons cannot be visualised on their own since they need some data to be realized; the exception is the Coastlines icon which when visualised produces a geographical map plot with the corresponding coastline plotting specification.

Table I-1 : Summary of Visual Definitions, applicable data or plot component and plot type.

Visual Definition	Data Type or Plot Component	Plot Type
Annotation	Plot Annotations	Any
Contouring	Gridded scalar data (incl vectorial modulus) in GRIB format	2-D Contouring and Shading, Isotachs, Numbers and Symbols at Grid Points
Wind Plotting	Gridded vectorial data in (U,V) or (r, θ) representation (wind components, wave direction and intensity). GRIB format	2-D arrow, WMO wind flag or streamline plots
Observation Plotting	Meteorological observations in BUFR format: SYNOP, AIREP, SATOB, DRIBU, TEMP, PILOT, SATEM, ECMWF Feedback	WMO-standard meteorological observations
Symbol Plotting	Any irregular (scalar/vectorial) point data in a geographic or XY axis system. Geopoints or NetCDF format	Text string, marker, number, wind arrow at a location (geographical, XY axis or paper)
Graph Plotting	Any (X1,X2/Y1,Y2) data, incl Lat Long cords	XY curves and scatter plots, bar charts, area-fill curves, trajectories
Image Table	Satellite data (continuous grid)	Grey or colour scale picture
Text Plotting	Plot Title	Any
Coastlines	Land and sea outline and shading, map grids and labels	Any geographical plot
Axis	Horizontal Axis (value and date) and Vertical Axis (value, model/pressure levels, linear/log)	Any 2-D axis system
Legend Entry	Legend format specification	Any
Drawing Priority	Order of visual definition plotting	Any
Overlay Control	Overplotting of fields	Any
Import Plot	Display of imported graphics files	JPEG/PNG imported files

Each Visual Definition applies to a type of data or provides a plot component such as coastlines or axis. Table I-1 lists the available visual definitions, the data they can be used with or the plot component they control and the type of plot they originate.

The View Concept in Visualisation

The final fundamental element in a visualisation is the **View**. A View is a user defined display specification which describes :

- **the plotting perspective** - i.e. what kind of plot to produce from the data : a geographical map, a cross-section, a vertical profile, ...
- **the plot structure** - for a map, the view specifies grids, coastlines, region limits and geographical projection; for a cross-section it specifies the coordinates of a transect line; for a vertical profile it specifies the coordinates of the point; for any plot using axis it specifies axis characteristics (min/max, tick marks, ...)

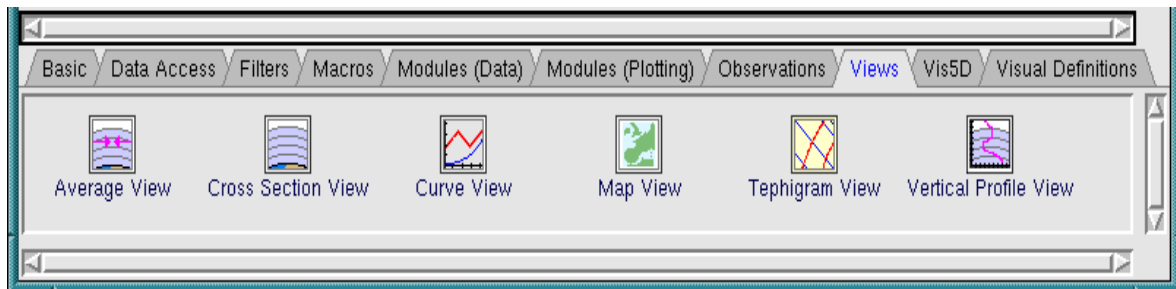


Figure I-40 : View icons available in Metview residing on the Views desktop drawer.

Table I-2 : Summary of Views and corresponding output examples.

View	Description	Parameters	Output Examples
MapView	Geographical area in a given cartographic projection	1. Bounding box (lat-long) 2. Projection	2-D Field Shading and Contouring, Observations, Wind Plotting
CrossSectionView	Cross section of the atmosphere, along a line defined on a lat-long grid	1. Transect line (lat-long) 2. Initial and final levels 3. Vertical axis (linear/log)	Cross Section
AverageView	Cross section of a zonal or meridional average atmosphere	1. Thin area (lat-long) 2. Initial and final levels 3. Vertical axis (linear/log)	Average
HovmøllerView	Hovmøller diagram	None	Hovmøller diagram
TephiView	Two drawing areas with the tephigram special coordinate system	1. Minimum and maximum T 2. Top and bottom pressures	Tephigram
VerticalProfileView	Axis system with height as the vertical coordinate and parameter value as the horizontal coordinate	1. Top and bottom pressures 2. Vertical axis (linear/log) 3. Point/area (lat/long)	Vertical Profile
CurveView	General XY axis system with user defined horizontal and vertical coordinates	1. Horizontal / Vertical Axis 2. Type (curve/bar/area) 3. Graph parameters: colour, thickness, line type, symbol	Curves (scatter, curve, bar, area), time series and trajectories (in a lat-long axis)
Satellite View	Earth disc with coast and gridlines in geostationary satellite projection	1. Satellite subpoint, subarea	Geostationary satellite image display
Import View	Graphics File Import	1. Positioning 2. Plot Border Specification	Embedded Graphics Files
Text View	Text Annotation Import	1. Positioning 2. Text Box Border Spec	Embedded Text
Empty View	Empty area for layout purposes	None	Empty space

As everything in Metview, views are specified by means of View icons - available ones are shown in Figure I-40. So, to produce a plot from data unit icons you need to define and supply a view icon. Even if you don't do it explicitly (e.g. when visualising a data unit directly), Metview will always provide a default view icon (which depends on the type of data unit icon to visualise). View icons

can be visualised on their own, the outcome being a display window with a blank plot of the type corresponding to the view.

For each type of plot you can prepare in Metview, there is a corresponding view, or put the other way around, if there is no view icon for a given type of plot then you cannot prepare such a plot in Metview - a descriptive list of the views available in Metview, with input parameters and typical output plot is shown in Table I-2.

This example layout divides a Display Unit in three Plot Frames, each using a different View : a Cross Section View, a Vertical Profile View and a Map View in a 3x1 arrangement

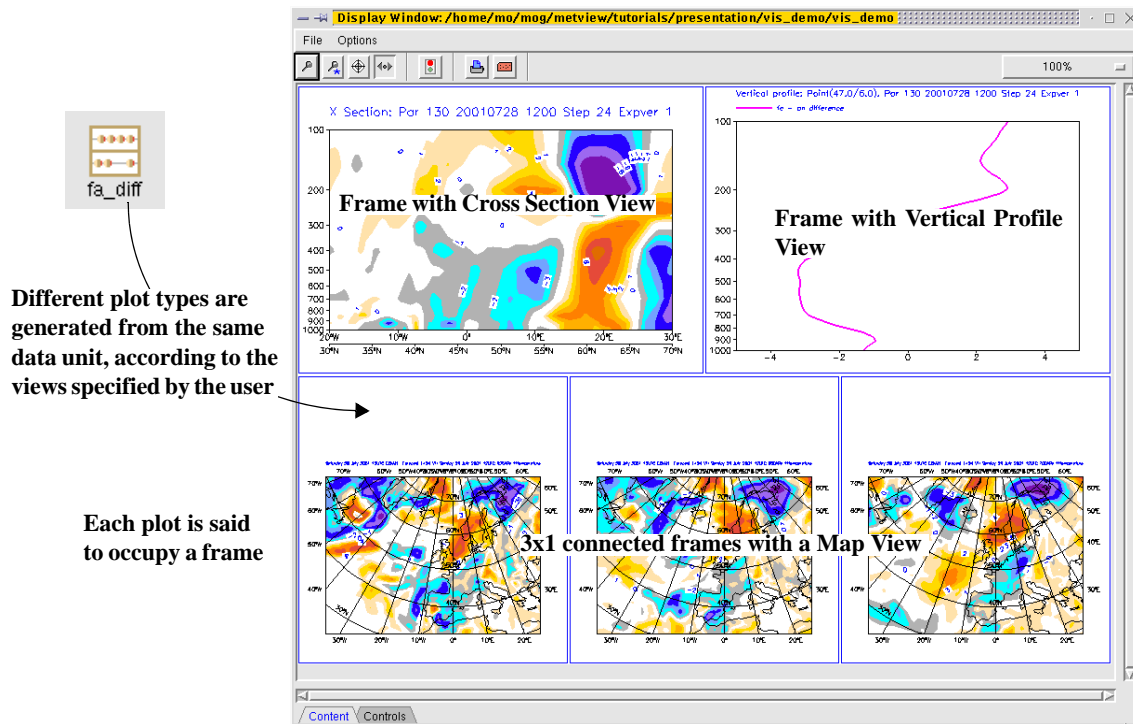


Figure I-41 : The View concept at work - using different types of views within a layout arrangement, you can produce different types of plots from the same data unit.

Since the View determines the type of plot you get from a given data unit, you can prepare *different plots from the same data unit*, simply by visualising it through different views (provided the data unit is composed of suitable data). This is exemplified in Figure I-41 showing a layout visualisation with three different plots generated from the same data unit icon (Simple Formula icon computing multi-level forecast-analysis differences) through different views : a Cross-Section View in the top-left frame, a Vertical Profile View in the top-right frame and a Map View in the bottom half frame.

Finally, you should note that you may obtain exactly the same plot by visualising a suitable data unit through the required view or by providing the data unit as input to an application which produces an output data unit that leads to the plot. If obtaining a cross-section plot from a data unit of a set of multi-level model fields (as a file or database retrieval), you get the same result by :

- visualising the data unit through a *Cross-Section View*
- using the same data unit as input to the *Cross-Section Application* and visualising the application icon or the NetCDF file it can produce as output

Direct Visualisation

This is a fast visualisation, carried out straight from a data unit icon, i.e. any icon that returns data - a MARS Retrieval icon, a data filter icon (GRIBFilter, ObsFilter), a GRIB file icon, a Formula icon, a Macro icon or one of the application icons (e.g. Potential Temperature).

In a direct visualisation, you do not prepare any layout before visualising the data icon - a default layout with a default view is used instead. This default arrangement is an A4 landscape display unit with a Map view. For a *direct visualisation* of a data icon, simply :

- click-right the data icon and choose the option **Visualise**
- supply appropriate Visual Definition icons

This visualisation is "direct" as you are not required to specify anything about how you want to plot the data. Since no plot instructions are provided, Metview will identify the type of data returned/generated by the icon and will assign to it the corresponding view; all non-specified elements are assigned built-in system defaults.

However, the interactive display window provides a great degree of flexibility and through its functionality and through icon drops you can modify extensively the initial result to tailor it to your requirements. In particular you can :

- change the details of the default view, e.g. by selecting a new projection and/or a new sub-region
- provide an entirely different view (if the visualised data unit is suitable), by dropping a view icon you have already - this means you can transform your plot from a map to a cross-section
- drop other visual definition icons to modify the look of the plot

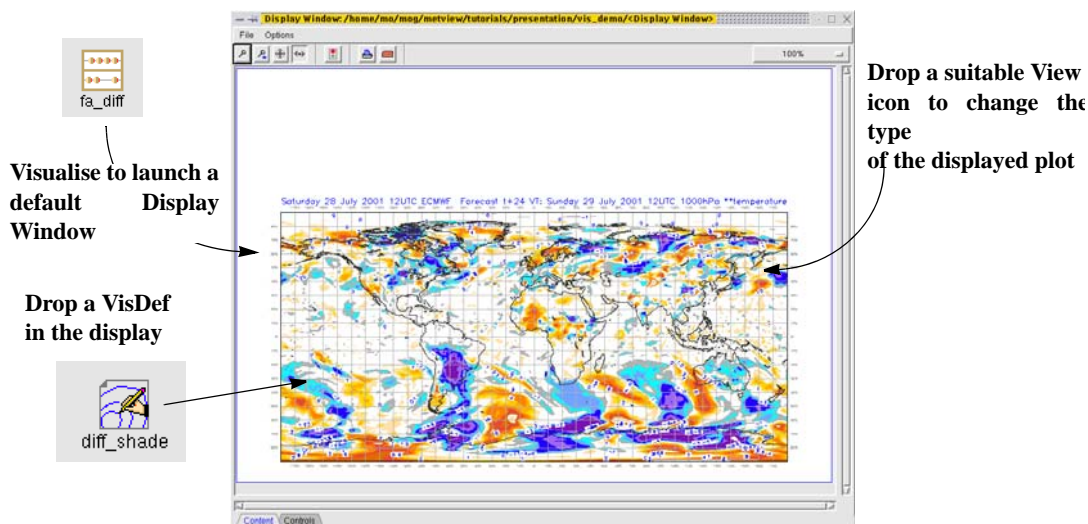


Figure I-42 : Schematic representation of a Direct Visualisation. If you only need a single-plot visualisation, this is the fastest way to have it up and running

The major limitation in a direct visualisation is that you are confined to use single frame, single plot layouts as the layout is the one thing you can never modify interactively. All things considered, a direct visualisation is most useful for informal visual data exploration (zoom in and out, point value examination, animations, change of projections and views) well served by the default layout.

Layout Visualisation

Layout visualisation must be used for any visualisation involving more than one plot. As the name implies, the user must first create a layout for the plots by means of a Display Window icon. The data and visual definitions are provided after the display window is up and running on screen. This visualisation mode provides full user control over what is displayed and allows complex visualisations to be built.

Figure I-43 shows the main steps of a Layout visualisation process and how the different intervening icons contribute to the final visualisation product. The steps are as follows :

- Preparation of a layout - using the Display Window icon editor, specify a display size (A4, A3 or custom) and orientation (landscape or portrait), and subdivide the display unit in a number of *plot frames* following any required arrangement - this process is detailed in "The Display Layout" on page I- 71
- Definition of the views - decide the types of plot you need and specify their details by means of View icons to be embedded in the Display Window icon editor
- Visualising data - execute the Display Window icon and then drop data and visual definition icons in the display window frames

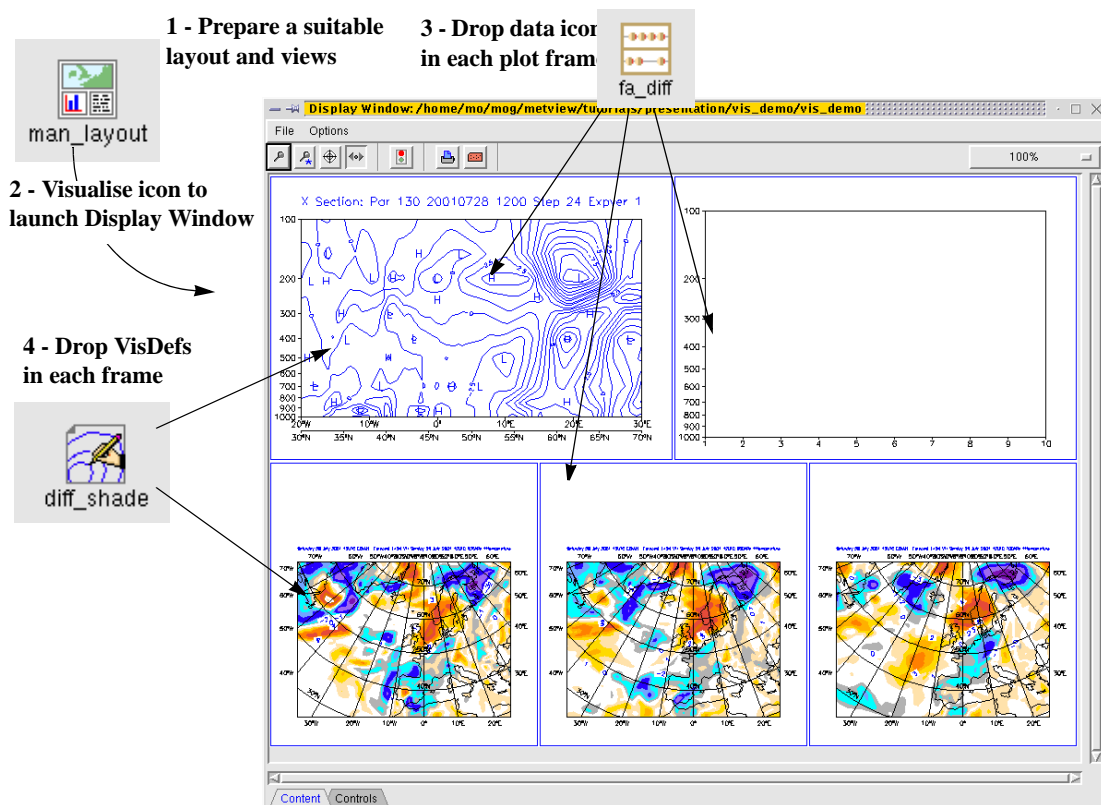


Figure I-43 : Schematic representation of a Layout Visualisation. A multi-view visualisation like this one requires some work before it is up and running. However, you can store all the layout elements as templates for later re-use with no need to rebuild everything from scratch

From here, you can also modify extensively your visualisation, changing visual definitions, replacing the current views, providing new data, modifying the geographic elements of the visualisations (projections, transect coordinates, etc).

In the earlier stages of using Metview, some effort is required before you have a visualisation up and running, but this improves as you acquire practice and specially if you make a consistent use of templates and build up your own library of predefined Display Window icons.

THE DISPLAY LAYOUT

Overview of Layout Design

Designing a layout is the crucial first step in data visualisation. Metview provides an interactive graphical layout editor which is the editor of the Display Window icon. This editor allows you to specify :

- A display size, specified as paper sizes A4, A3 or user custom size
- A display orientation specified as landscape or portrait
- A display sub-division into *plot frames*, either regular MxN or arbitrary

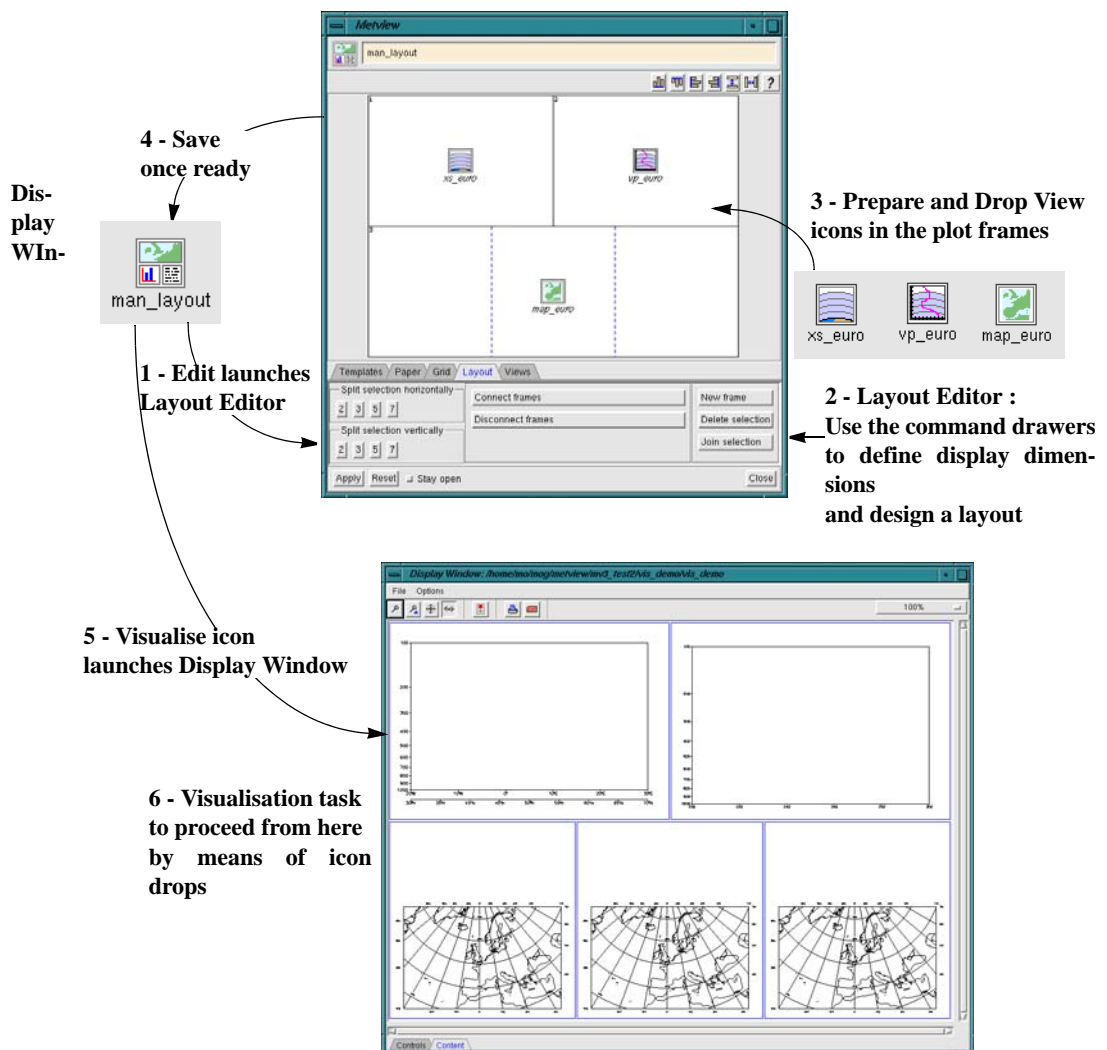


Figure I-44 : Example of a layout preparation as the basic step for a complex visualisation. The end result of the layout is a Display Window icon which will provide an empty display unit for the user's visualisation. This will proceed by means of icon drops

Each display subdivision is said to be a **Plot Frame** - hence the layout is composed of frames. Once the layout is defined you have to drop suitable View icons in each frame. When you are happy with the layout you have designed, you save and close the Display Window editor. When executed or visualised, the icon will launch a display unit on screen with the defined layout and views. The process is shown in Figure I-44. The layout building process is described in detail in the remainder of this chapter.

The Display Window Icon Editor

Metview allows users to mix different plots in the same display unit (on-screen window, graphics file or paper) and arrange them in any way they see fit. A layout for these plots can be prepared quickly and effectively through the interactive layout editor of the Display Window icon.

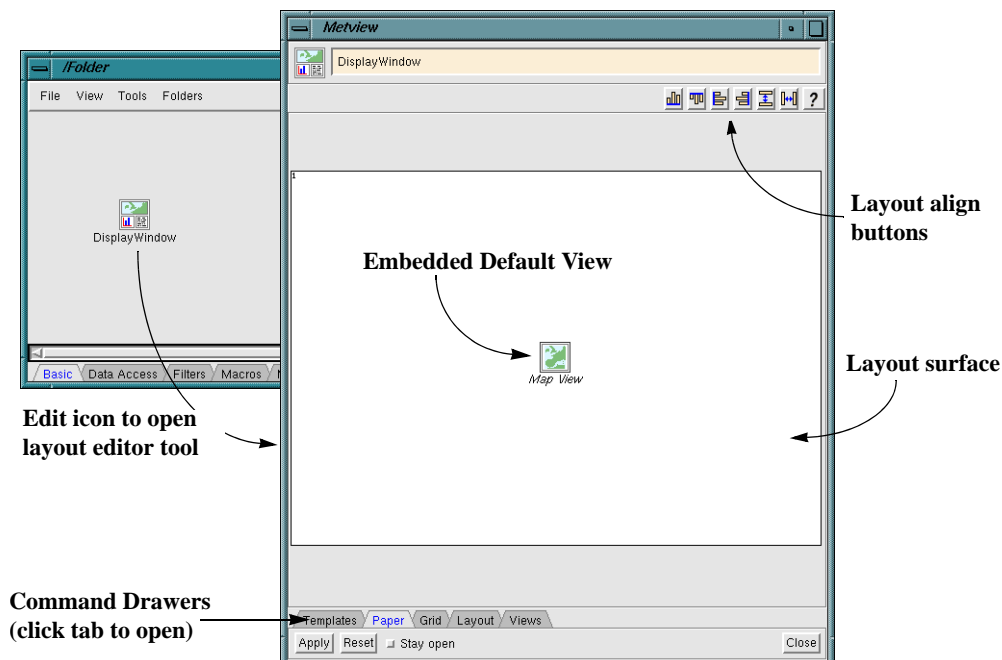


Figure I-45 : The default Display Window editor with main features highlighted.

The Display Window icon differs from the other Metview icons in that it has a graphical interactive editor window, as shown in Figure I-43 and Figure I-45. The default layout surface in the editor is a blank white rectangle with A4 landscape proportions.

The editor has a number of tabbed drawers, each containing commands to define and set different aspects of the layout :

- **Paper** drawer : Size and orientation of display
- **Layout** drawer : Subdivision of the display area
- **Grid** drawer : Fine control of plot positioning
- **Templates** drawer : Storage of predefined layouts
- **Views** drawer : Storage of predefined View icons

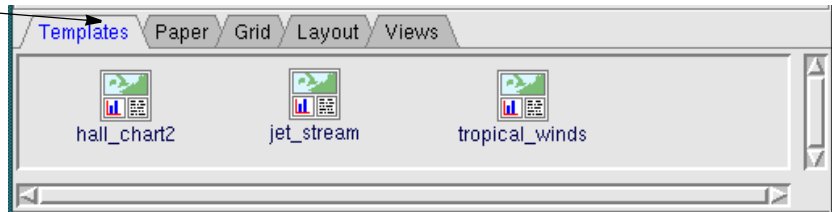
Figure I-46 shows the full complement of command boxes available in the Display Window editor. The controls inside each tabbed drawer provide the required functionality for layout design - to access a tabbed box, simply click-left on its tab. Click again to close it. The next sections cover the ways in which you can use these commands to design a display layout.

For all drawers :

Click tab to open, Click again to close

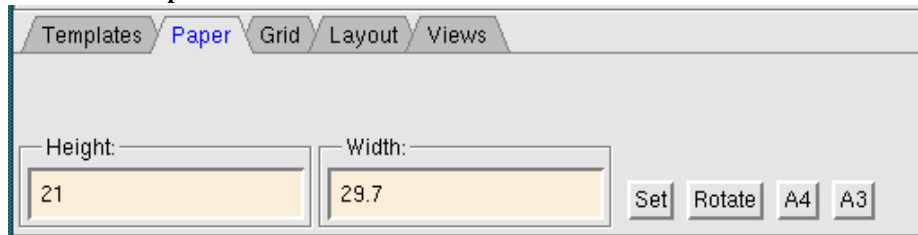
Drag Display Window icons into main window to copy their layout

Templates Drawer

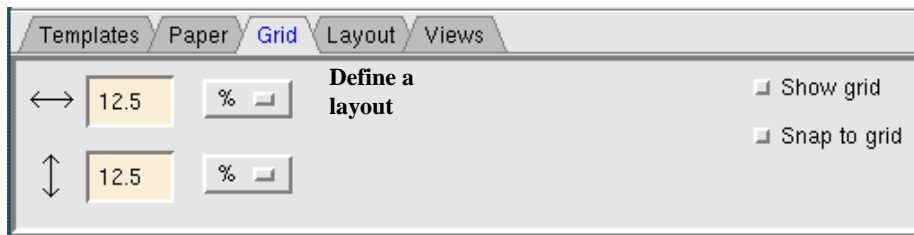


Define paper size and orientation

Paper Drawer



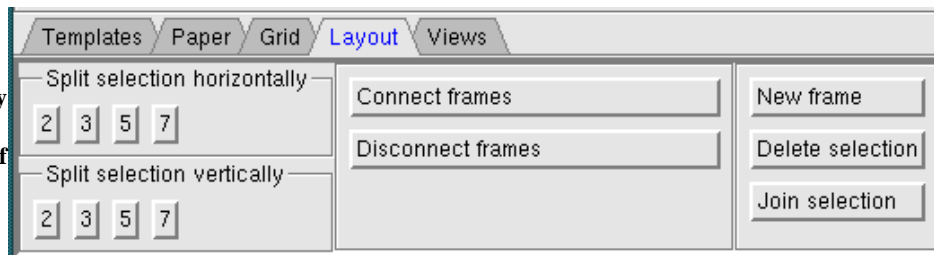
Grid Drawer



Enable grid features

Layout Drawer

Subdivide display into a number of frames



Control frames

Connect and disconnect frames

Views Drawer

Drag View icons into main window to embed in the layout frames

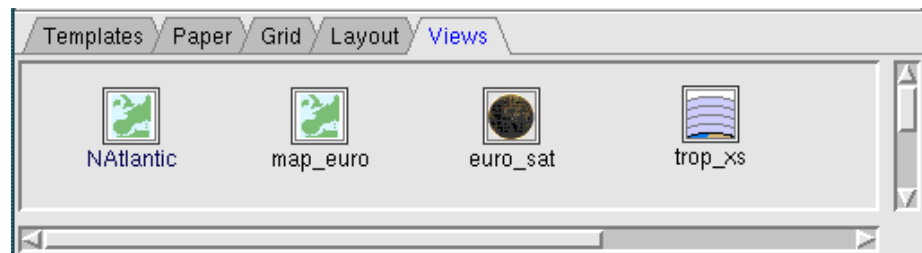


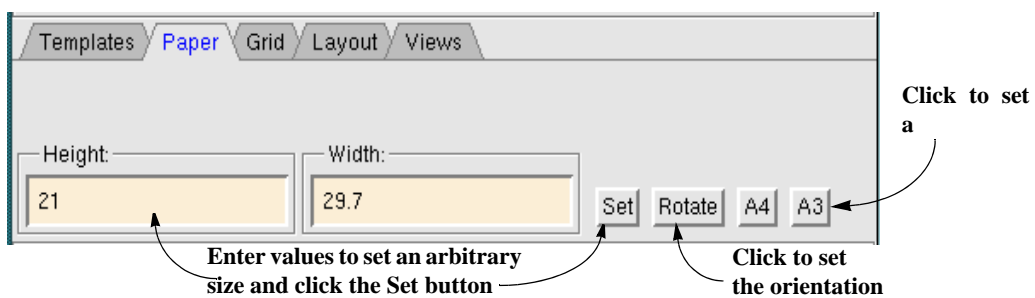
Figure I-46 : The Command boxes of the Display Window editor and associated functions. See text for details.

Preparing Display Layouts

Setting size and orientation

To specify the display dimensions :

- Click the Paper tab to open the box
- Choose a pre-set dimension - A3 or A4 - by clicking the respective button or
- Enter values in cm in the Height and Width boxes and click the Set button

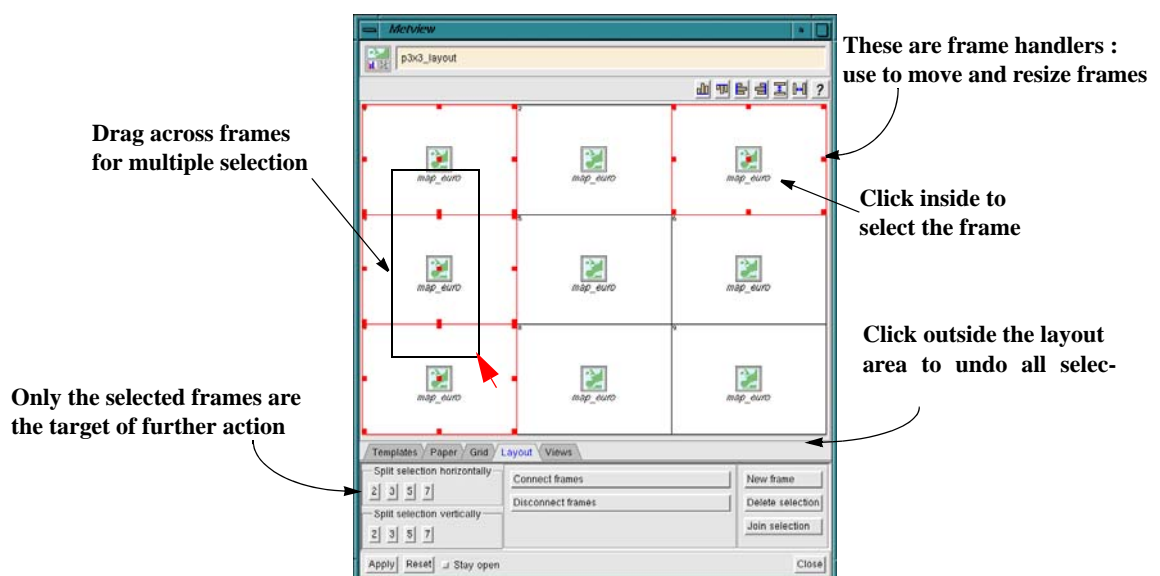


To set the orientation :

- Click the Rotate button to toggle between portrait and landscape.

Selecting frames

Once the size and orientation of the display area are defined, further layout design operations - subdividing, un-dividing (unifying), re-sizing, moving - apply to one or more **frames**. However these operations only apply to frames which have been Selected.



To select :

- a particular frame : click inside it
- contiguous frames : click inside the first one and drag across to the last
- non-contiguous frames : click one frame and Shift-click the other frames
- all frames : as contiguous or simply type Ctrl-A

To un-select :

- all frames, except a particular one : click inside the frame you want to keep
- a particular frame : Shift-click inside the frame that you want to un-select
- all frames : click just outside the display area

When a frame is selected it is highlighted in red - note also the presence of squares in each frame corner, each mid-point of the sides of the frame and the centre of the frame. These are **frame handlers** and you use them to re-size the frame (side and corner handlers) or move the frame (centre handler) within the display area.

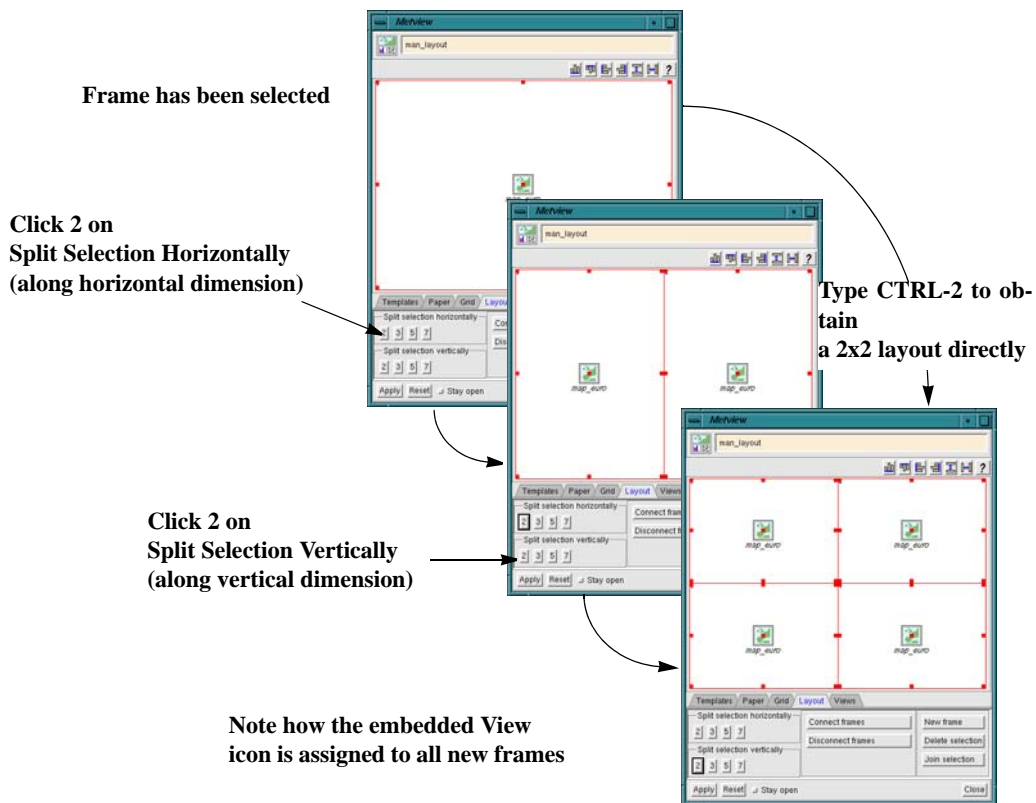
Sub-dividing frames

Frame sub-division can be carried out by means of mouse or keyboard.

- Select the frame(s) to be sub-divided (see above)
 - Open the Layout box and click one of the numbered buttons. The editor has separate sets of buttons for vertical and horizontal sub-division - Split selection horizontally and Split selection vertically. Note that *horizontally* and *vertically* refer to the frame dimension along which the division is done
- or
- Type Ctrl-N, where N is the number of subdivisions you need. Note that this keyboard sub-division applies to *both* page dimensions, i.e. it will subdivide the selected frame in N×N frames. You must select only a single frame for this type of subdivision. If you type Ctrl-Shift-N you obtain the same result but with gaps between the subdivided frames

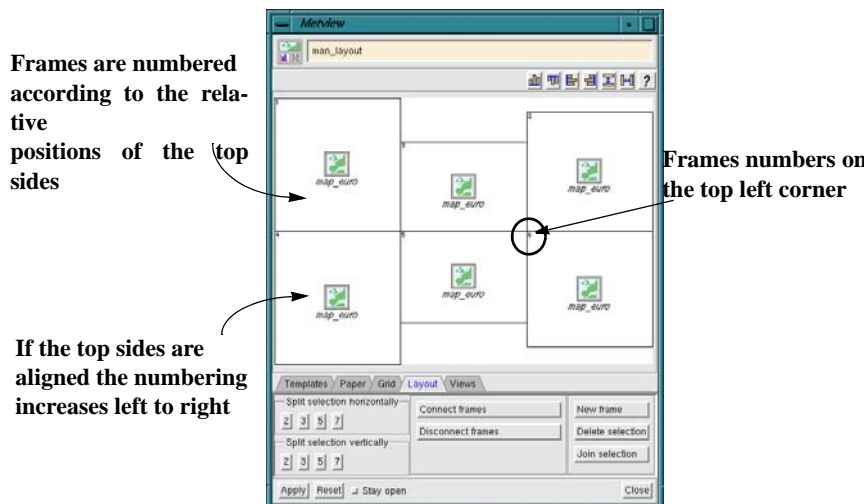
It is very easy and straightforward to generate display areas with regular subdivisions - you can go from a blank display to a N×M display in 4 mouse clicks : one to open the Layout box, one to select the area, one to sub-divide in N along a direction, another to subdivide in M along the other dimension.

Note that if you have a view icon residing on a page, sub-dividing this page will create a number of smaller pages each with the same view icon, offering a quick way to create postage-stamp layouts.



Frame numbering

Once you have a multi-frame layout you will notice that each frame is assigned a number residing on its top left corner. The frame numbering starts at the top-left most frame and increases rightwards and then downwards, i.e. given two frames with their top (left) sides aligned the leftmost (resp uppermost) one has the lower number.



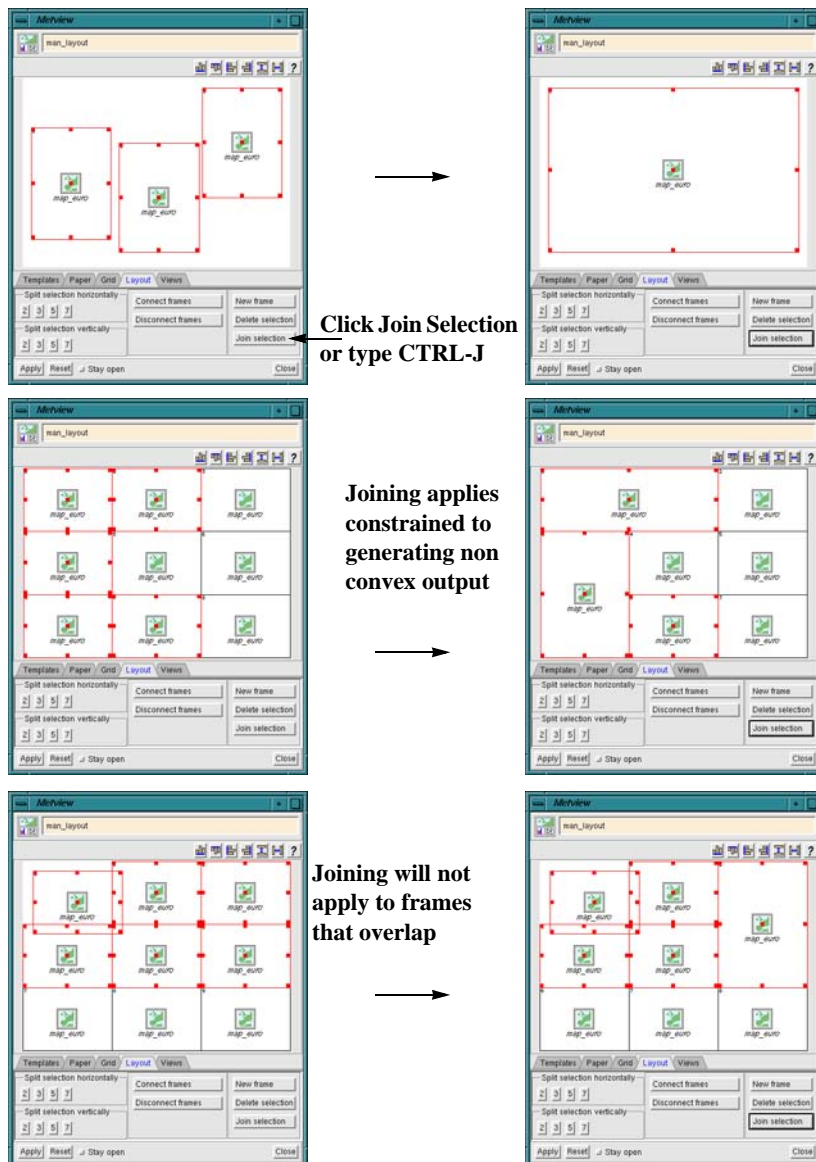
Note that the *numbering represents order of drawing*, which means that in case of frame overlap, *the frame with the highest number is drawn on top*.

Joining frames

Joining is the reverse of sub-dividing frames and can be carried out by means of mouse or keyboard :

- Select the frames you need to join
- Click the Layout tab to open the box and click the Join Selection button
or
- Type Ctrl-J on the keyboard

Joining acts on selected frames



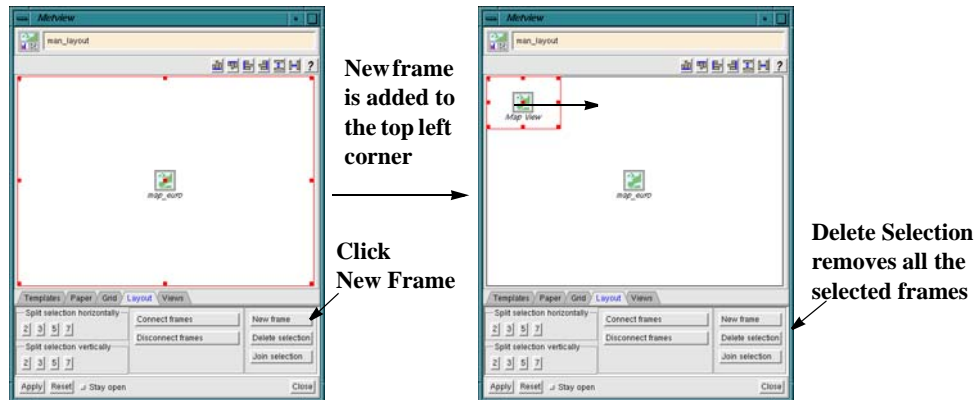
When you join frames of different dimensions and positioning, the dimensions of the resulting single frame will maximally enclose the original frames. Joining only works if the frames to be joined do not overlap and if the resulting frame does not intersect /overlap other frames. Joining will always apply as far as possible, hence depending on the arrangement of selected frames, you may end up joining only part of the original selection of frames.

Adding and deleting frames

To add a new frame :

- Click the Layout tab to open the box and click the New Frame button

This new frame has a small size and is always added to the top left corner of the display unit. You have to move it where you need it and re-size it according to requirements.



To delete existing frames :

- Select the frames you need to delete
- Click the Layout tab to open the box and click the Delete Selection button

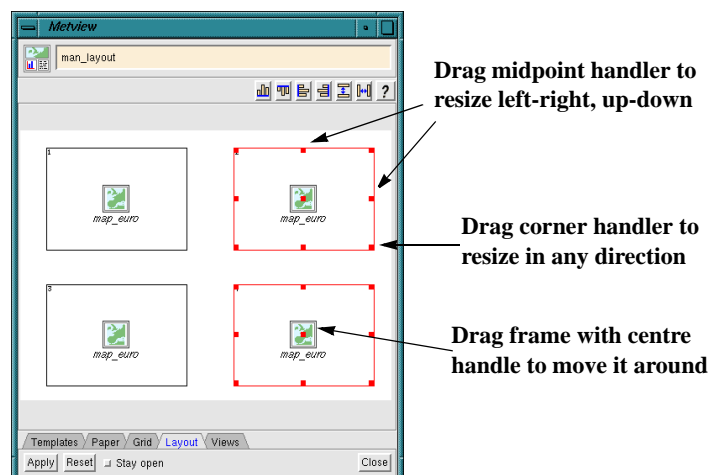
Deleted frames leave a blank space behind. You can delete each and every frame, if you want. If you are left with no frames at all, you can add a new one. Otherwise, on saving and re-editing the icon a single frame will always be provided.

Moving and re-sizing frames

To move frames within the display area :

- Drag the frame centre handler in the required direction

Moving and re-sizing will affect all the selected frames.



To re-size frames :

- Drag one of the frame's corner or side handlers in the required direction

Side handlers allow only up-down and left-right resizing while corner handlers allow re-sizing in any direction. Frames can overlap other frames - the one with the highest number is plotted on top.

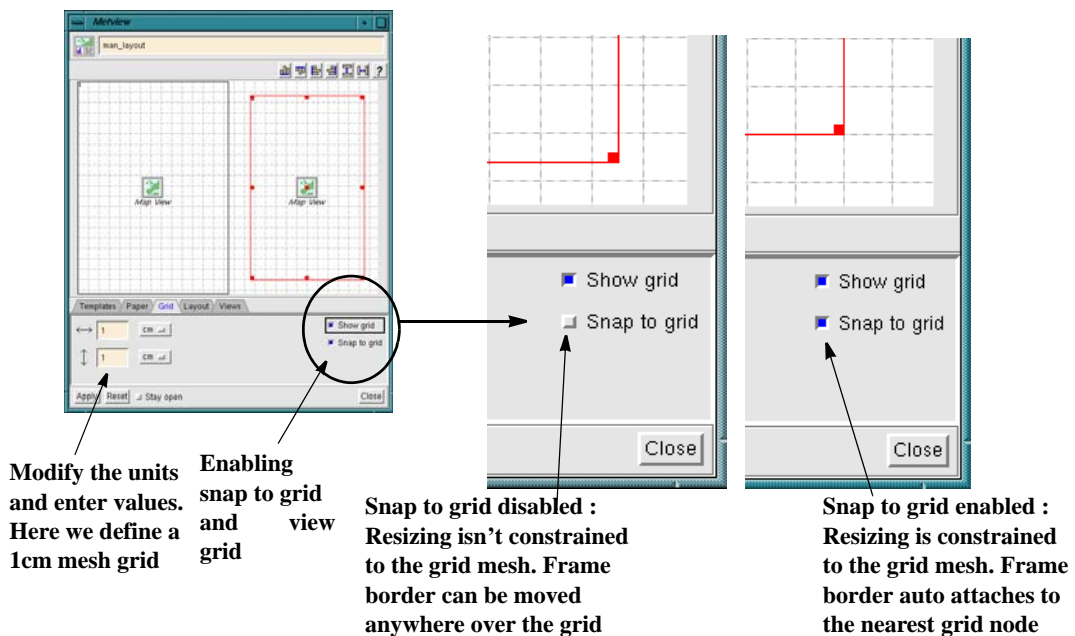
Note that if you have several frames selected, re-sizing and moving one of them will affect all those selected - you can use this to move or re-size entire sets of frames.

For fine control of the frame movement and re-sizing, use the **Snap to grid** functionality (see below).

Using grid control in frame moving and re-sizing

The **Grid** functionality is used when you need to resize frames to particular sizes or to move frames to specific relative positions within a display unit.

The idea is to define a grid of a given mesh size defined in cm or percentage of display dimensions to help you position the frame. The finer the grid mesh, the more accurate the positioning. Enabling the *snap-to-grid* feature makes the positioning very precise.



To use the grid features :

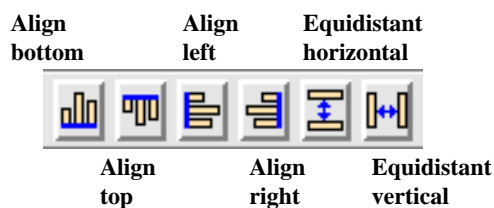
- Click the Grid tab to open the box
- Select the units for the grid dimensions. By default this is expressed as percentages of the display dimensions, otherwise select centimetres
- Type in the required numbers for the grid dimensions and press Enter
- Click the Show grid check button to make the grid visible
- Check the Snap to grid check button to enable the snap-to-grid feature.

With this feature enabled, you only need to resize the frame approximately, as it will automatically attach itself to the nearest grid node

Frame alignment

If you need to keep accurate alignment between frames you can use the **alignment buttons** - these are placed on the top right of the editor window. To use them :

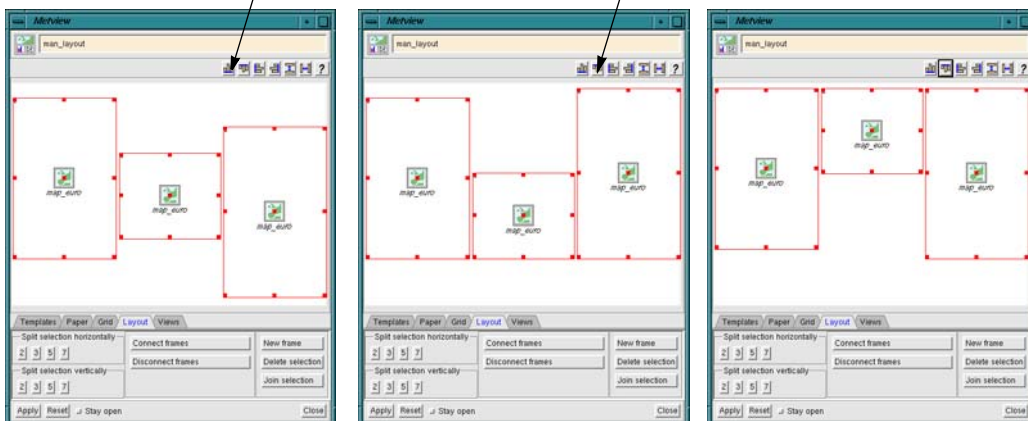
- Select the frames whose sides you want to align
- Click the button that provides the required alignment



Initial arrangement
with frames selected

Click align-bottom

Click align-top



The alignment reference frame is the one with the lowest number

The alignment buttons can align the left, right, top and bottom sides of any number of pages. You can also align the pages in such a way that their centres are (vertically or horizontally) equidistant one from the other.

For all alignment modes one of the frames is chosen as the reference one. The other frames are aligned relative to the reference frame. The reference frame is the one with the lowest number.

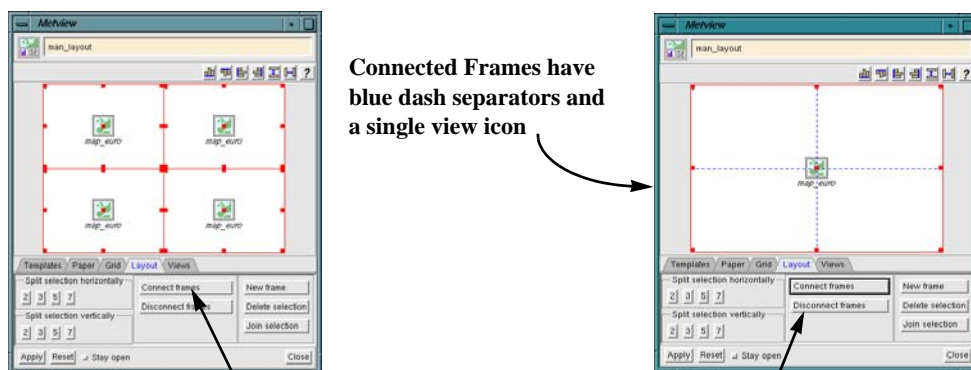
Connecting and disconnecting frames

A layout is composed of a number of frames. Each frame is assigned one View icon to specify the type and characteristics of the plot to be produced from a given data unit. A data unit can give rise to multiple plots, such as :

- a dataunit with n forecast fields viewed through a Map View originates n 2D geographic plots
- a data unit of m ensemble members at n levels through a Cross-Section View will generate m cross-section plots

When you generate multiple plots you may need to show a given number out of the total number simultaneously in a given layout. Or showing all of them in one go, maybe as a regular m*n arrangement. By default, frames only allow a single plot to be displayed at a time. Here we see how you can make a frame display more than one plot at a time and how you can specify an arbitrary layout for these plots.

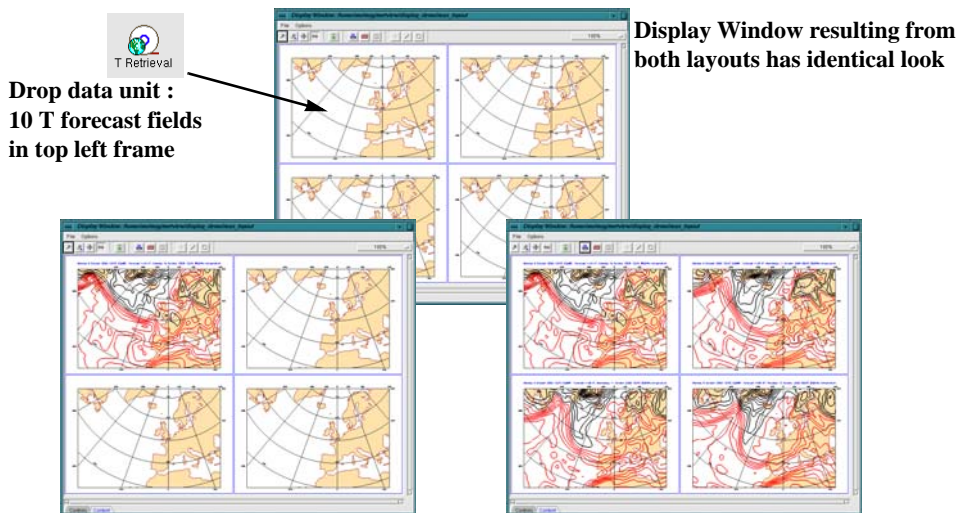
The solution is simply to define a given layout of frames in the usual way and then to *connect* some or all of these frames. When you connect frames they behave as a single frame as far as accepting data is concerned - it is just that the plots that are generated are distributed through the connected frames in sequence. Hence a set of connected frames can only have a single type of view.



Devise a layout, then select frames

Click Connect Frames

Disconnect Frames returns to previous status



Drop data unit : 10 T forecast fields in top left frame

Display Window resulting from both layouts has identical look

Independent frames : data is plotted on the frame where it was dropped. The frame has a stack of plots, that users can scroll up and down. Other frames can remain empty or accept other unrelated data units

Connected frames : data is plotted over all the frames, including when users scroll through the data stack. Frames are always full and always use the same visual definitions

The steps required to create a set of connected frames are :

- Create a layout of independent frames using the techniques described above in this section, e.g. a plain $m*n$ layout from a single original frame as in "Sub-dividing frames" on page I-75
- Select all the frames to connect and click the Connect Frames button in the Layout command box

This generates a set of connected frames easily identified by their blue-dash separators. Individual connected frames are not accessible for operations of any kind - a set of connected frames behaves as a single frame for layout purposes. However, note that a connected frame set can not be subdivided any further, but you can re-size and move it at will - when re-sizing, the component frames will resize proportionally.

If you have a set of connected frames you can return it to its unconnected status simply by:

- Selecting the connected frame set and clicking the Disconnect Frames button in the Layout command box

Specifying views

To specify which view is used in a frame, you have to assign an icon of the required view to that frame. This is done simply by dropping the required view icon inside the frame.

You can drop view icons at any stage of the layout design. If you subdivide a frame which has a view icon, *all the frames* resulting from the subdivision *will have the same view* icon. This can greatly speed up the process of creating single view $N*M$ layouts - first assign the view icon to the original frame and then subdivide it.

If you want to change the view currently assigned to a frame, simply drop the new view icon in its place. If you need to leave a page blank use an Empty View icon.

Empty space

There are two ways to create an empty space within a layout. First, you can simply move and resize the frames around the area you wish to be empty. The second method is to create a frame and drag an Empty View icon onto it. Since the second method will result in an empty frame with a border, the two methods are not identical.

Example Layout Preparation

Here we show how the layout of the sample visualisation in Figure I-41 was prepared.

- 1 - The starting point is to create a Display Window icon on a desktop and *Edit* the icon

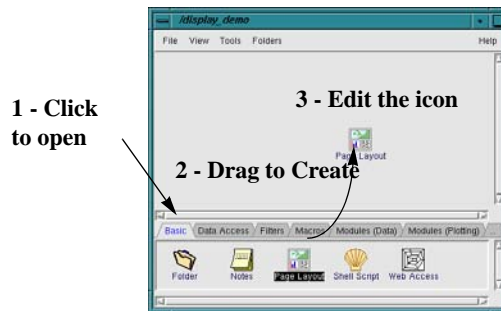


Figure I-47 : Explicit creation of a Display Window icon.

2 - The editor opens with a default layout (in this example) of an A4 Portrait page with a Map View. On the interactive icon editor, click the Paper tab to select the paper size and dimension. Change it to A3 Landscape, by clicking the button A3 and then the button Rotate. The icon may be renamed within the editor's icon identification field.

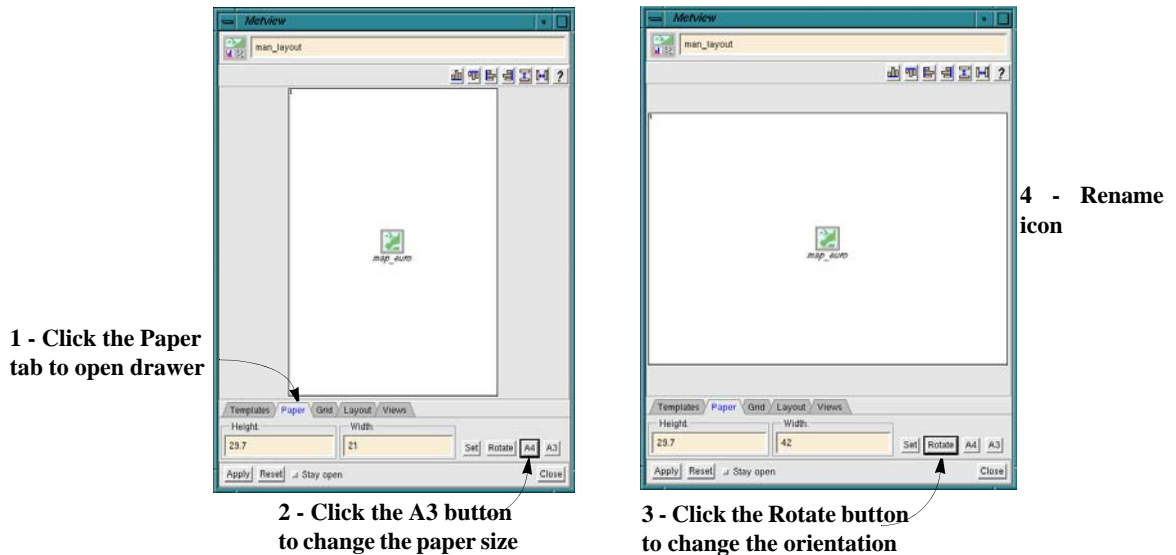


Figure I-48 : Changing the display unit dimensions and orientation

3 - Now define the subdivisions of the display unit - Click the Layout tab to open the drawer. The desired division can be appreciated in. The easiest way is to :

- split display in two along the vertical dimension (Split Selection Vertically)
- split top half in two along the horizontal dimension (Split Selection Horizontally)
- split lower half in three along the horizontal dimension
- connect the lower group of three frames

The sequence of operations above illustrates how you can easily define a complex layout interactively.

Once the layout is defined you have to supply the views to go with it. If not done yet this requires you to set up and prepare their icons. The view icons are simply dropped into the desired page. The work is saved with an Apply in the layout icon editor. The process is shown in Figure I-49

The icon can then be used in visualisation work - execute or visualise the icon to launch the display window. To visualise data, simply drop data icons on the display frames; visual definition icons can be dropped together with or following the data icons. The process is shown in Figure I-43 on page I-69.

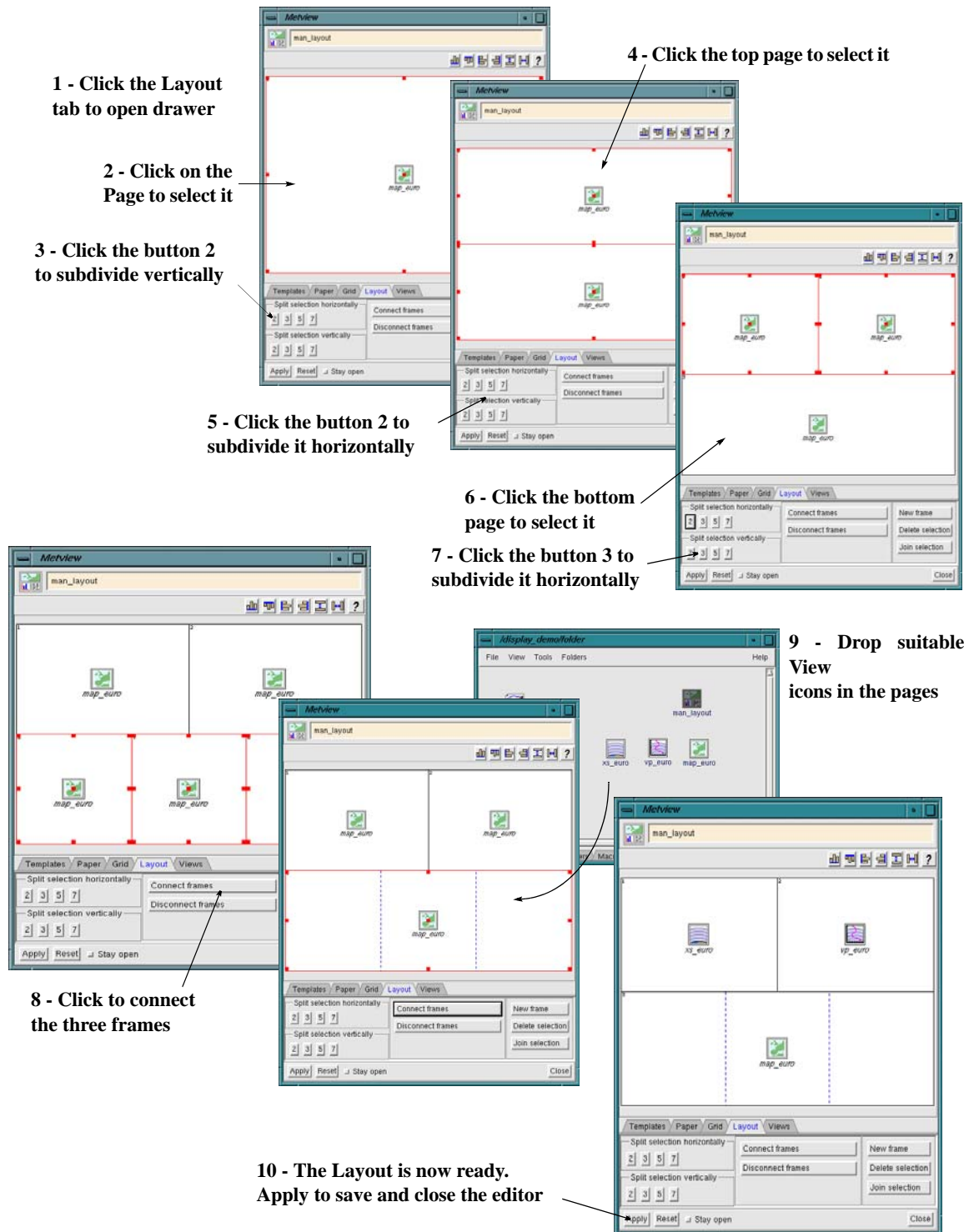


Figure I-49 : Steps required to create an example multi-view layout

VIEWS AND VISUALISATION

Role of Views in Visualisation

The role of Views in the visualisation of data can be easily understood by recapping the concept of View presented in "The View Concept in Visualisation" on page I- 65 :

A View is a plotting specification which controls :

- the type of plot to produce from the data unit (map, cross-section, ...)
- the geographical elements of the plot - this includes projection and area coordinates, transect line, averaging area or point coordinates; also axis limits and specification, geographical grid spacing, land and sea shades.
- the overlaying of multiple data units - controlled by a set of overlay switches
- the plot positioning within the display unit (e.g paper sheet) - specified as an X and Y off-set and lengths expressed as a percentage of the display unit dimensions
- the plot border specification - on / off plus colour, format and thickness

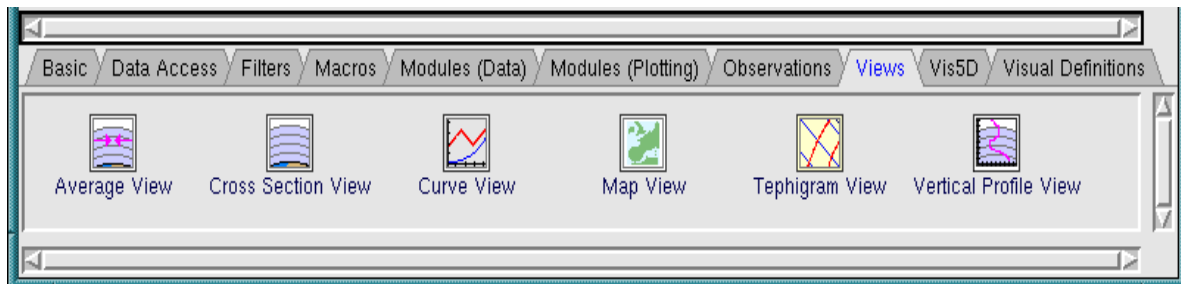


Figure I-50 : View icons available in Metview residing on the Views desktop drawer.

Each type of plot you can produce in Metview has an associated view. If there is no view for a particular type of plot, this plot cannot be produced in Metview. Since the view determines the plot type, you can produce several types of plot (map, cross-section, vertical profile, ...) from the same data unit, provided the data icon returns suitable data.

Each view is represented by a View icon whose input parameters specify the controls listed above. The input parameters as listed in the editors of the view icons can be divided into different groups :

- Geography and computation parameters
- Overlay Controls
- Plot positioning and dimensions

These are detailed in the next subsections.

Geography and computation parameters in views

The first group of input parameters sit at the top of the editor and are specific to each different view - they determine the plot type by specifying input such as :

- geographical coordinates of areas, lines and points, projection types
- axis specification (by means of numerical values and axis icons)
- computation details (direction of averaging, log or linear pressure axis)

The listing of such parameters for each of the available views is detailed next. In all the view editors, the geographical coordinates are entered by the users either directly or using an interactive map based geography (help) tool - this tool is fully described in "The Geography Tool" on page I-91. Note that there are views for which this type of input element is not applicable, e.g. the Text View.

Map view

In Map views, users can provide the following elements :

Coastlines - to specify the map coastline format (on/off, resolution, style, colour and thickness), the map grid format and spacing, the map label format and the presence/absence and colour of the land and sea shades.

Map Projection, Hemisphere and other projection details - you can choose between cylindrical, polar stereographic, Mercator, Aitoff and Lambert projections, ocean section and none; hemisphere (north and south) and vertical longitude only apply to polar stereographic projection; vertical longitude is the longitude at the center of the plot and allows users to rotate this projection. The remaining parameters apply to the Lambert projection. Note that it is up to the user to provide a valid set of projection parameters. If the desired parameters are not already known, then the Geography Tool should be used. See "The Geography Tool" on page I-91 for details. If the supplied projection parameters are not valid, Metview will use default values and issue a warning message in the Messages window (see "The Message Window" on page I-126).

Area - this is the region to be plotted; enter the coordinates of the bottom left latitude and longitude and the top right latitude and longitude of the region as a sequence of number separated by forward slashes.

To ease the specification of these geographical elements (particularly in the case of the polar stereographic projection), the Map view editor has a geography specification help tool - this is an interactive tool that enables a quick and easy selection of projections and areas. See "The Geography Tool" on page I-91 for details.

Cross-Section view

In the Cross-Section view, users can provide the following elements :

Line - to specify the start and end coordinates (latitude, longitude) of the transect line along which the cross section will be derived. Users can specify the transect line coordinates interactively by means of the geography tool associated with the Line input parameter field. This is similar to that of the Map view but is more restricted in its facilities, see "The Geography Tool" on page I-91.

Top pressure and Bottom pressure - to specify the values appearing as the end points of the pressure (vertical) axis in a cross-section plot.

Pressure Level Axis - the axis can be set to Linear or Logarithmic.

Average view

In the Average view, users can provide the following elements :

Area - to specify the coordinates of a rectangular area (top left latitude, longitude, followed by bottom right) over which the average will be derived. Users can specify the area coordinates interactively by means of the geography tool associated with the Area input parameter field. This is similar to that of the Map view but is more restricted in its facilities, see "The Geography Tool" on page I-91.

Direction - to specify the direction along which to carry out the averaging (NS or EW).

Top pressure and **Bottom pressure** - to specify the values appearing as the end points of the pressure (vertical) axis in the plot.

Pressure Level Axis - the axis can be set to Linear or Logarithmic.

Vertical profile view

In the Vertical Profile view, users can provide the following elements :

Input mode - to specify whether to derive a profile from point values or area averages

Point - coordinates of point (latitude, longitude) at which values will be derived (by interpolation)

Area - coordinates of area (top left followed by bottom right latitude and longitude), over which vertical profile values will be derived (by averaging)

Users can specify the point or area coordinates interactively by means of the geography tool associated with the Point or Area input parameter field. This is similar to that of the Map view but is more restricted in its facilities, see "The Geography Tool" on page I-91.

Top pressure and **Bottom pressure** - to specify the values appearing as the end points of the pressure (vertical) axis in the plot.

Pressure Level Axis - the axis can be set to Linear or Logarithmic.

Pressure Axis and **Vertical Axis** - to specify formatting details of the pressure (vertical) and value (horizontal) axis by means of two Axis visual definition icons

Data overlay control in view icons

When visualising several data units, possibly involving several parameters, forecast steps, levels, etc, in the same display window frame, you need some rules to define which (if any) of the data is *plotted together in the same plot* (this is known as **data overlaying**). The choices the users have are to :

- Never overlay the different data units
- Always overlay the different data units
- Control the data overlaying by means of **overlay settings**

The use of overlay settings works in the following way :

- The user selects one or more attributes of the data (e.g. forecast verification date, model or pressure level) to act as overlay settings.
- Only the data units with the same value for the selected attributes are overlaid; otherwise they are plotted separately.

The overlay controls (never / always / use settings / attributes to use as settings) are specified in the View icons by means of an embedded Overlay Control icon (see "Overlay Control" on page III-237).

See full details in "Data overlay rules and overlay controls" on page I- 116ff.

Plot positioning and dimensions

The Views also control the plot positioning and dimensions within the display unit (on-screen display window plot area or paper sheet). By **plot** we mean strictly the data plot, *not including title, legend or any other elements (plot border)* - see the next subsection for information on title and legend positioning.

The positioning is controlled by a set of parameters which define X and Y offsets and lengths, expressed as a *percentage of the plot surface dimensions* (e.g. A4, A3 or custom). These parameters are present in all view icons under the same names :

Subpage X Position - horizontal offset position of the plot relative to left margin (expressed as a percentage)

Subpage Y position - vertical offset position of the plot relative to top margin (expressed as a percentage)

Subpage X Length - horizontal length of the plot starting from the horizontal offset position (expressed as a percentage)

Subpage Y Length - vertical length of the plot starting from the vertical offset position (expressed as a percentage)

Figure I-51 shows an example of the effect of varying these dimensions.

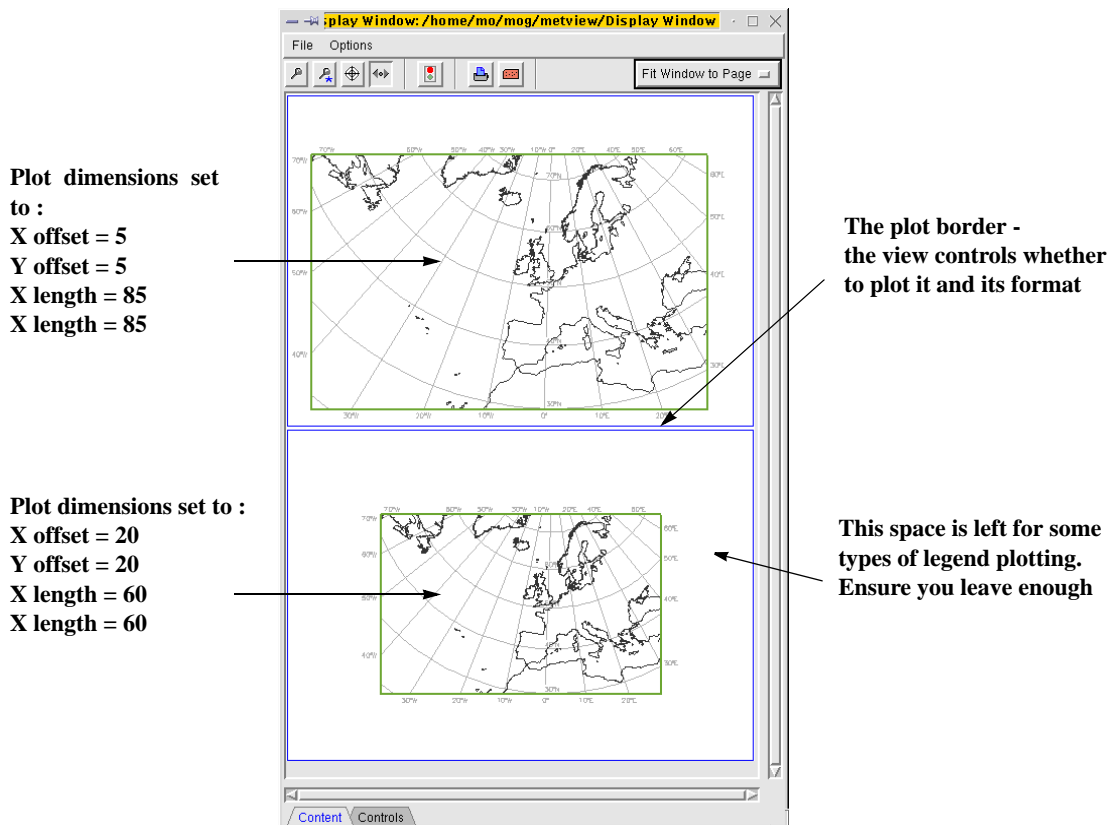


Figure I-51 : Effect of different offset and dimensions on a visualisation. The two views in usage differ only in the indicated offsets and dimensions. Also indicated is the plot border

An important factor to remember is that the offsets and dimension control the plotting of the legend and titles of plots - Metview will try to fit the legend and the title in the space left after the plot has been drawn. *If you have set the plot dimensions and offsets in such a way as to leave too little space on the right side and top side you may not get a legend or title plotted even if you explicitly specified it.*

Plot borders

The final aspect controlled by the views is the plotting of borders around the plot. All the view editors contain a set of input parameters which specify whether to plot this border and its format :

Page Frame - Toggle plotting of plot border (page frame) on or off

Page Frame Colour / Line Style / Thickness - the format details of the plot border line

The default plot border in a display window is shown in Figure I-51.

The Geography Tool

The **geography tool** is - a *graphical map-based interface which allows users to specify interactively geographical input such as coordinates of areas, lines or points as well as projection types*; it basically consists of a map window (heavily based on the visualisation display window) with a set of tools that enable users to :

- select a geographical projection out of cylindrical, polar stereographic (north and south) and Mercator; in some flavours of the geography tool the only available projection is the cylindrical (e.g. line selection in the cross-section view)
- select an area, line or point over a map
- zoom in the displayed map to ease fine area/line/point coordinate extraction
- customize some aspects of the geography tool display

You can see a geography tool depicted in Figure I-52 and Figure I-53. In Metview, there are two situations when you need to use the geography tool and each has a different way to launch it :

- When specifying the geographical input to a View icon for further use - in this case, you launch the geography tool from the help button associated with the Area, Line or Point input parameter of the Map, Cross-Section, Average, Hovmöller Data and Vertical Profile View editor icons - see Figure I-52.
- When modifying the geographical details of an on-going visualisation, e.g. changing the projection and region, or the transect line - in this case, you launch the geography tool using the Change Geography option in the Display Window frame menu - see "Changing the display geography" in "Using the Display Window" on page I- 102.

When the geography tool is up, you use its functionality to specify your geographical elements interactively. When you are happy with the result, you close the geography tool and the settings you derived are passed back to the icon editor input parameters or to the on-going visualisation depending on how you called it.

The geography tool comes in different flavours (Area, Line and Point geography tools), depending on which type of view you are editing or using in the visualisation in question:

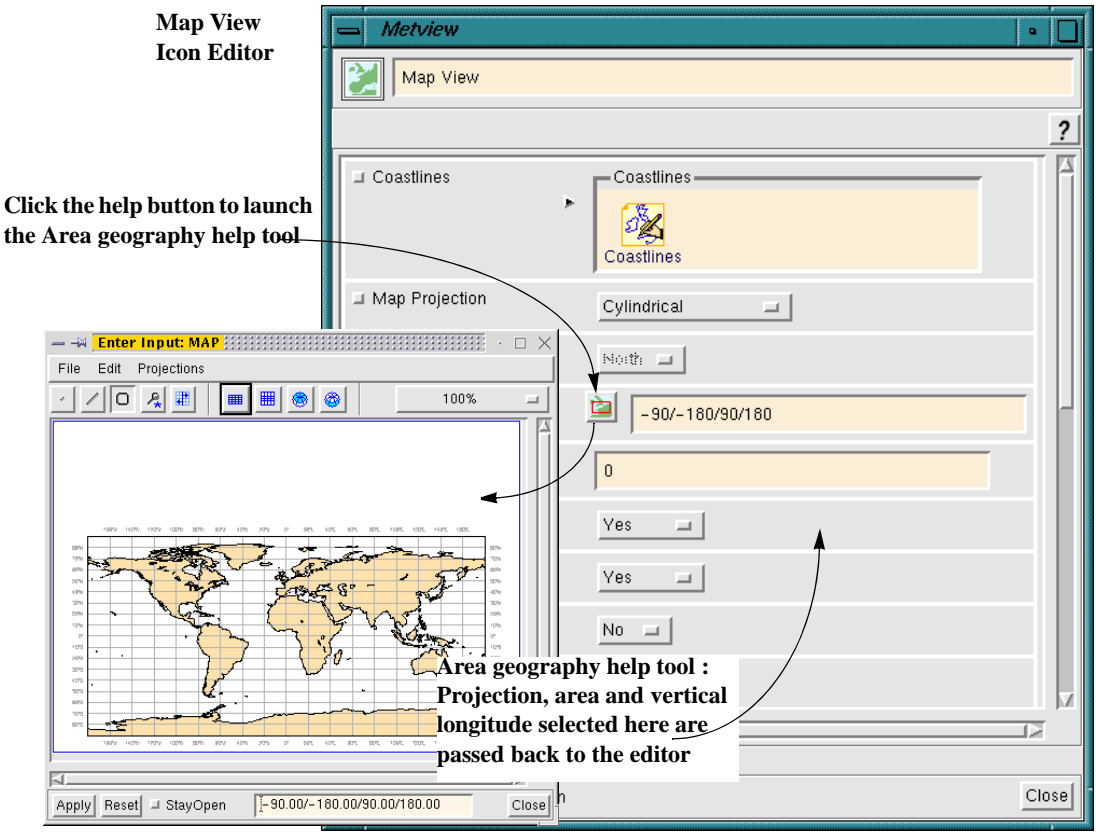
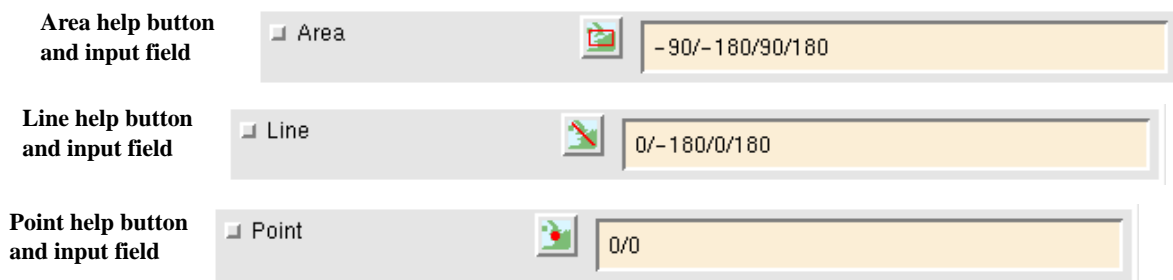


Figure I-52 : Launching the geography tool from the Area help button within a Map view icon editor. Also shown are close-ups of the Area, Line and Point help buttons

- Map View - the Area geography tool is launched, enabled with the full complement of the projections available in Metview - cylindrical , mercator, polar stereographic north and south. Choice of region, projection and central longitude
- Cross-Section View - the Line geography tool is launched, enabled only with the cylindrical projection. Choice of transect line and central longitude
- Average View - the Area geography tool is launched, enabled only with the cylindrical projection (input areas for Average must be rectangular on regular lat-long grid). Choice of region and central longitude
- Vertical Profile View - the Point tool or Area tool is launched depending on the type of vertical profile specified. In both cases the tool is enabled only with the cylindrical projection. Choice of point or region and central longitude

Figure I-53 shows a typical default Geography tool. As you can see the differences between the flavours listed above reflect themselves in the number and type of command buttons which are available or not. Unavailable command buttons are shown greyed out in the user interface.

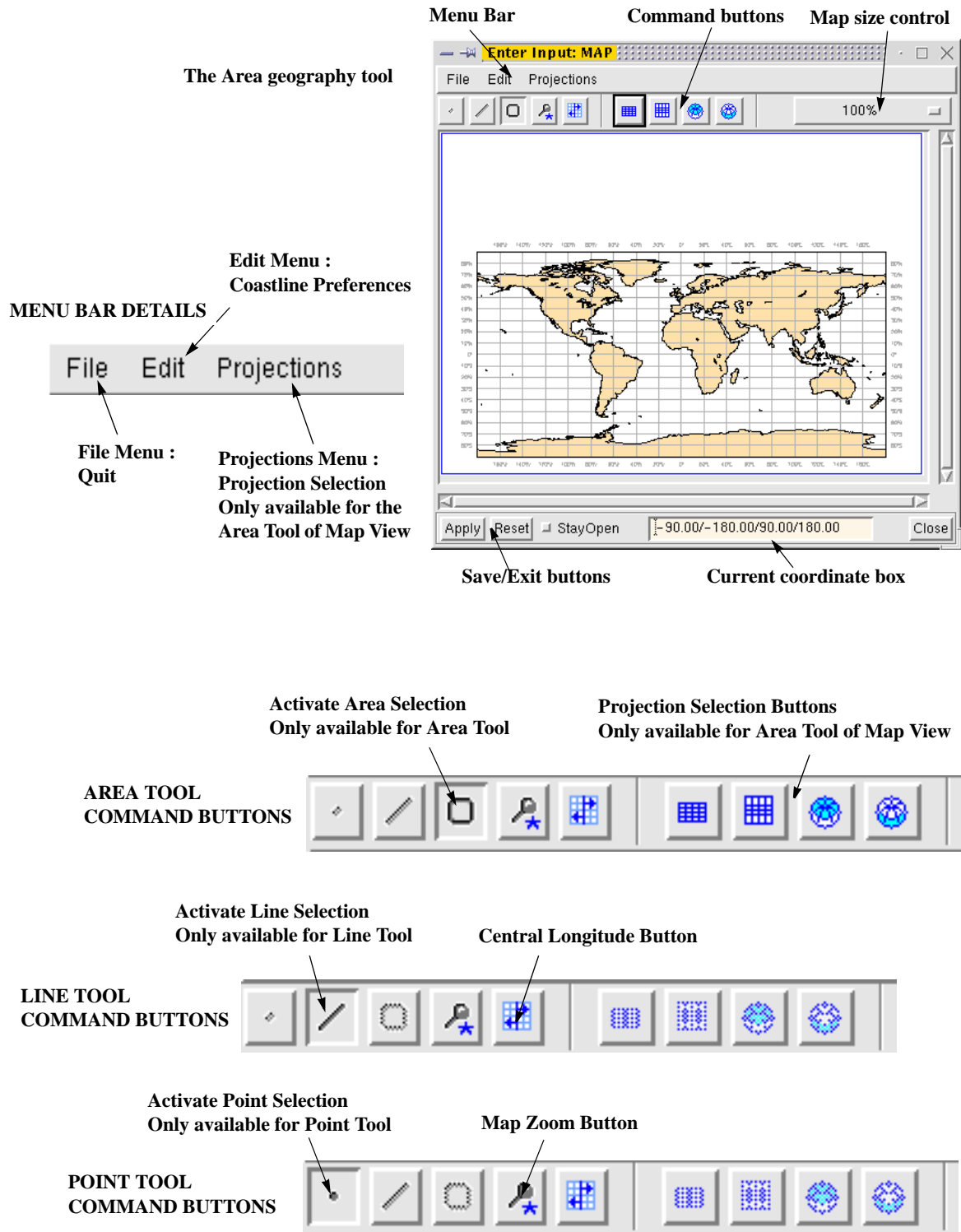


Figure I-53 : The Geography tool. The general look is identical for all flavours - Area, Line and Point - but each has different command buttons available.

Working with the Geography Tool

Overview

The geography tool is used to specify geographic elements, i.e. coordinates of areas, lines and points as well as projection types and details. A typical geography specification session would have the following steps :

- selection of a projection type (unless restricted to cylindrical)
- selection of a vertical longitude to centre the display map
- zooming in an area to ease specification of small areas or short lines
- selecting your area, line or point
- saving and exiting

Upon saving, the elements you specified are passed back to the view editor or to the current visualisation depending on the working context of the geography tool.

Zoom

The geography tool is equipped with a Zoom facility. This exists for all the flavours of geography tool. *The zoom facility is not for area extraction!* Its purpose is to help in the selection of areas, lines or points. A typical usage of the zoom is when users need to select a fairly small area - in this case a prior zoom eases the selection. To use the zoom :

- click-left on the zoom button to activate the zoom
- drag across the displayed map to select a rectangular region
- the selected region replaces the previous displayed map

You can keep on zooming at will. All the zoom actions are stored in memory and you can travel up (and down) the zoom sequence or clear it altogether :

- click-right on the map and select one of the options Zoom Out or Zoom In, from the pop up menu. The option Zoom Reset clears the zoom sequence and returns you to the default map

When you are finished with the zoom action you can move on to the next action (e.g. select an area, change projection, ...) by clicking its command button - this de-activates the zoom and activates the new action.

Choice of projection

Metview provides the following projections in the geography tool :

- Cylindrical
- Polar Stereographic (north and south)
- Mercator

To select a projection, simply click-left its button - this displays the global map of that projection. You can then select a new vertical longitude, zoom in on the displayed map or select a geographical

element. To discard any changes it is enough to click the projection button again to return you to the global area.

Vertical longitude

The vertical longitude selection allows you to rotate your map. This works by displaying the map in such a way that the longitude line you select for vertical longitude occupies the centre of the plot.

- On the cylindrical or Mercator projection this allows you to rotate the map longitude-wise, e.g. to centre the display on the Pacific ocean instead of the Atlantic. If you need to select an area or transect line across the Pacific, you have to specify a suitable vertical longitude
- On the polar stereographic projection it rotates the map around the pole, e.g. to make North America occupy the lower centre of the display instead of Europe

To select a vertical longitude :

- activate the vertical longitude selector with a click-left on its command button
- the cursor changes to a gun-sight shape; click-left on the map at any location along the required longitude coordinate
- repeat to adjust further, if required

Once you're happy with the result, move on to the next action by clicking-left on its command button. Note that when you select a vertical longitude the displayed map is returned to the global area of the chosen projection. So *always zoom after selecting the vertical longitude*, otherwise the zoom is lost.

Extraction of geographical elements

Once you are satisfied with the currently displayed map (maybe after a projection selection, vertical longitude selection and zoom action) you can proceed to selecting the geographical element you require :

- click-left the button corresponding to the geographical element you require, i.e. one of point, line or area; which buttons are available depends on the flavour of geography tool you are currently working with
- carry out the selection - drag a rectangle across the display map to select an area, drag a line to select a line transect, or click to select a point.

The defining coordinates of whatever element you are selecting will appear in the coordinate box below the display map

You can also enter specific coordinates in the coordinate box - simply type them in and the geographical element will be drawn on the display map. You can use this to make small adjustments to a mouse-cursor based selection or to enter a geographical element directly - in this case you may be using the geography tool simply to evaluate the relative positioning of your choice.

Customising the geography tool

You can customise the look of the display map by selecting the Coastline Preferences (single) option in the Edit menu. This launches the editor of the Coastlines icon which is used in the display

map. You can introduce any required changes to this coastline icon which will then be used in the display map, e.g. introducing a land (or sea) shade colour, the grid spacing and colour, etc.,

DISPLAY WINDOW FUNCTIONALITY

Introduction

The Display Window is Metview's on-screen interactive visualisation tool. With this visualisation tool you can :

- Magnify and zoom in on the visualised data
- Extract point values from displayed data fields
- Scroll through, browse and animate a set of plots
- Print some or all of the plots
- Translate the visualisation into a macro program

The visualisation display mimics a paper sheet by default (this is modifiable) and the visualisation is WYSIWYG. You are also allowed to change most visualisation elements except for the display layout and the intervening data units.

Users obtain a display window on screen through a direct or a layout visualisation of data unit(s). As an example, the layout visualisation work outlined in Figure I-43 leads to a final result shown in Figure I-54. This represents the Metview Display Window with its main features and components highlighted :

This chapter presents a summary of the functionality available in the display window.

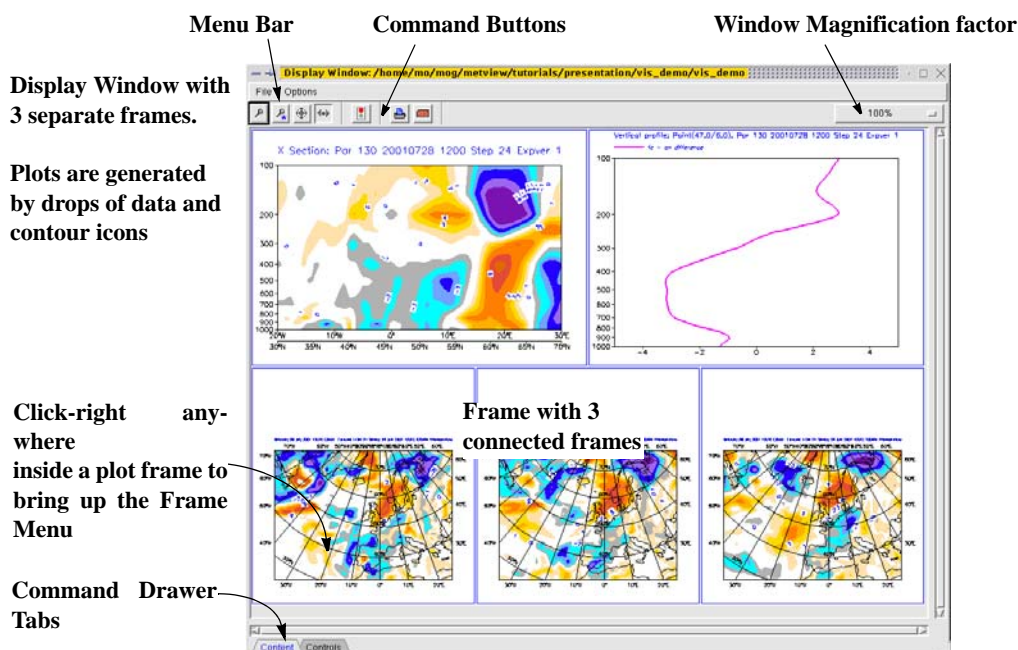


Figure I-54 : An example visualisation in a display window. The components of the display window are highlighted - see text for details.

Functionality Overview

Menu bar

File

Print - Launches the print editor menu. Users can set options for printing the current visualisation according to requirements. Clicking Apply on the editor starts the printing process - "Printing a visualisation" on page I- 112.

Generate Macro - This option creates a Metview macro program which will reproduce the current visualisation. The macro is created in the same folder where the display window icon resides. The macro has the default name PlotPageContentsN, where N is an ordinal number (the number of macros created in the current session) - see "Converting a visualisation to a macro program" on page I- 111.

Quit - Exits the display window

Options

Printer Preferences - Launches the print editor menu but users can only set printing options, not launch a printing action - clicking Apply saves the settings for the next print action - see "Printing a visualisation" on page I- 112

Command buttons

Command buttons are placed right below the menu bar. They provide much of the interactive functionality of the plot page. These buttons allow you to :

- Magnify and zoom in on the visualisation
- Extract point values from displayed data
- Scroll up and down a stack of plots
- Toggle on/off the background plotting of data
- Print the visualisation
- Translate the visualisation into a macro program
- Define zoom factors for the whole display window

Full details can be found in "Using the Display Window" on page I- 101.

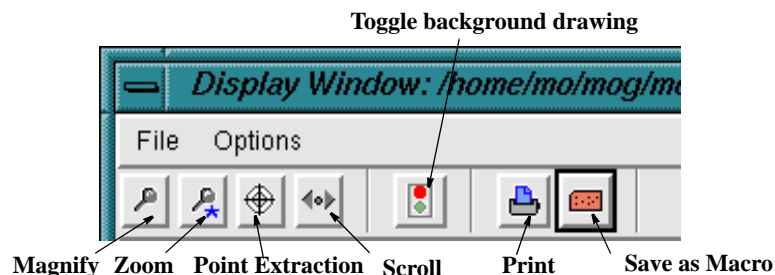


Figure I-55 : The display window command buttons

Frame menu

To obtain the frame menu simply click-right inside a display window frame. The menu has the following choices :

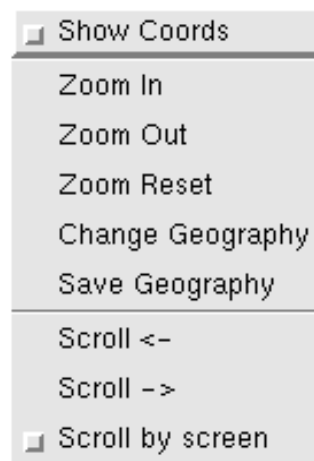


Figure I-56 : The frame menu options

Show Coords - Toggles the display of (lat-long) coordinates of the point where the cursor is. This is only available for Map View based plots - see "Point coordinates" on page I- 101.

Zoom In - Available only after you have carried one or more zoom actions. The successive zoom actions you may carry out are stored in memory and this command allows you to travel down (into) the zoom stack - see "Zoom" on page I- 101.

Zoom Out - Available only after you have carried one or more zoom actions. The successive zoom actions you may carry out are stored in memory and this command allows you to travel up (out of) the zoom stack - see "Zoom" on page I- 101.

Zoom Reset - Available only after you have carried one or more zoom actions. The successive zoom actions you may carry out are stored in memory and this command cleans the zoom buffer and resets the plot to its original status - see "Zoom" on page I- 101.

Change Geography - This launches a geography tool with the required functionality to modify the geography of your visualisation - e.g. changing the projection and area of a visualised GRIB field, changing the transect line of a cross-section, changing the coordinates of the point of a vertical profile. See "Changing the display geography" on page I- 102.

Save Geography - This option saves back to the original View icon on the desktop the current geography. This enables users to save details obtained interactively (e.g. by means of Zoom or Change Geography) and is available for direct visualisations, i.e. even when the View icon is a system default. See "Saving the display geography" on page I- 104.

Scroll <- - Scrolls backwards across a stack of plots (e.g. a set of GRIB fields) - see "Scrolling through a stack of plots" on page I- 107.

Scroll -> - Scrolls forwards across a stack of plots.

Scroll by Screen - when a frame has N subframes, you can scroll one by one or scroll N by N by setting this toggle on or off.

Contents drawer

The Contents drawer provides the user with a schematic display of all the icons involved in the visualisation and of their role and hierarchy within the visualisation layout - see Figure I-57. You can also remove, replace or edit most of the component icons giving you a very high degree of interactive control over the visualisation - see "Using the Contents Window" on page I- 121 for full details.

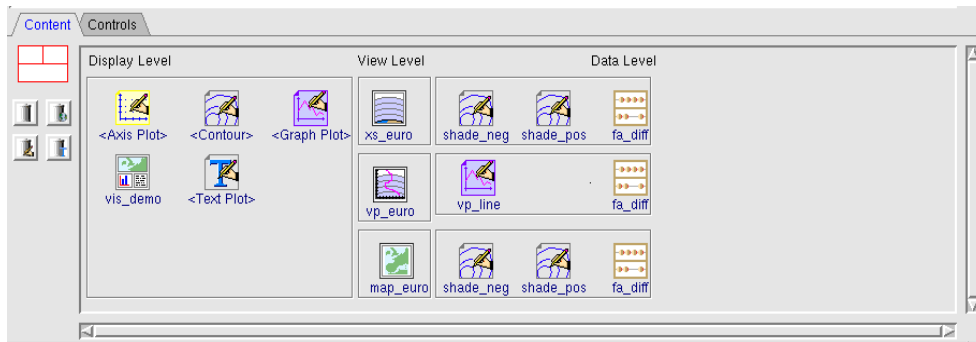


Figure I-57 : Opened Contents drawer for the sample visualisation in Figure I-54, showing the three level structure of a visualisation. Most icons can be individually manipulated and new icons added to the contents, allowing fine control of the visualisation

Controls drawer

This command drawer includes the functionality to control the display of plots in the window. It contains commands to :

- control the drawing of the plots
- browse through a stack of plots and select a particular one for display.
- control the animation of stacks of plots

Full details on the use of these control features can be found in "Browsing stacks of plots" on page I- 109 and "Animating stacks of plots" on page I- 110.

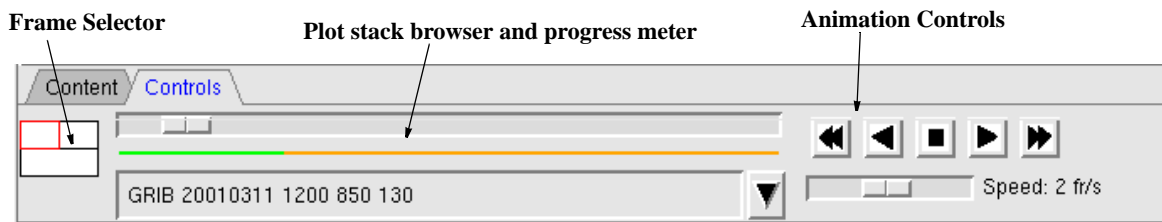


Figure I-58 : The Controls drawer with a frame selector, plot browser and animation controls

Using the Display Window

Here we present the display window visualisation tools around some plain visualisation examples. All the functionality presented here applies to more complex visualisations in exactly the same way. This section covers :

- Zooming in and out and magnifying visualised data
- Querying the visualisation, to get point coordinates and obtain the numerical field values at selected points
- Working with stacks of plots, i.e. scrolling, browsing and controlling the drawing of the plots
- Animating a stack of plots
- Obtaining a macro program that regenerates the visualisation
- Printing the visualisation

Point coordinates

To show map coordinates, click-right on a display frame and turn on the option Show coordinates in the frame menu. This is only available for plots using map views. The mouse cursor turns to cross-hair format and a small floating label with the coordinates at the centre of the cross-hair is displayed close to the cursor.

Magnification

To operate a magnification, click-left on the magnification button. Then drag-left the mouse cursor between two points within the display window frame - see Figure I-59. The selected area will be magnified in a separate floating window. The magnification is a purely graphical process and no recalculations take place : no new contour lines are added to the plot and characters are also magnified.

Magnification status stays active, so you can repeat the process as many times as required. Click-left on the button to toggle the magnification off. The floating magnification window has to be quit explicitly (File/Quit).

Zoom

To operate a zoom, click-left on the zoom button. Then drag-left the mouse cursor between two points within the display window frame - see Figure I-59. The selected area is displayed in the same display window frame. The visualisation is recalculated (note that it is not an instantaneous process) and this is different from the magnification in that new contour lines are calculated and characters are scaled - see Figure I-59 for a comparison between a magnification and a zoom. Zoom status stays active, so you can repeat the zooming as many times as required. Click-left on the button to toggle the zoom off.

When you do a series of successive zoom actions, the results are stored in memory. You can display again previous zooms using the frame menu (see "Frame menu" on page I- 99): its options Zoom In

and Zoom Out allow you to move back and forth along the zoom stack. Option Zoom Reset cleans the zoom buffer and resets the plot to its original dimensions.

You can use the zoom action to modify the geographic details in map view displays - if you want to save the current zoomed region back to the map view icon you can do so by selecting Save Geography from the frame menu. The geographic coordinate limits of the zoomed region will replace the current ones held in the map view icon in use.

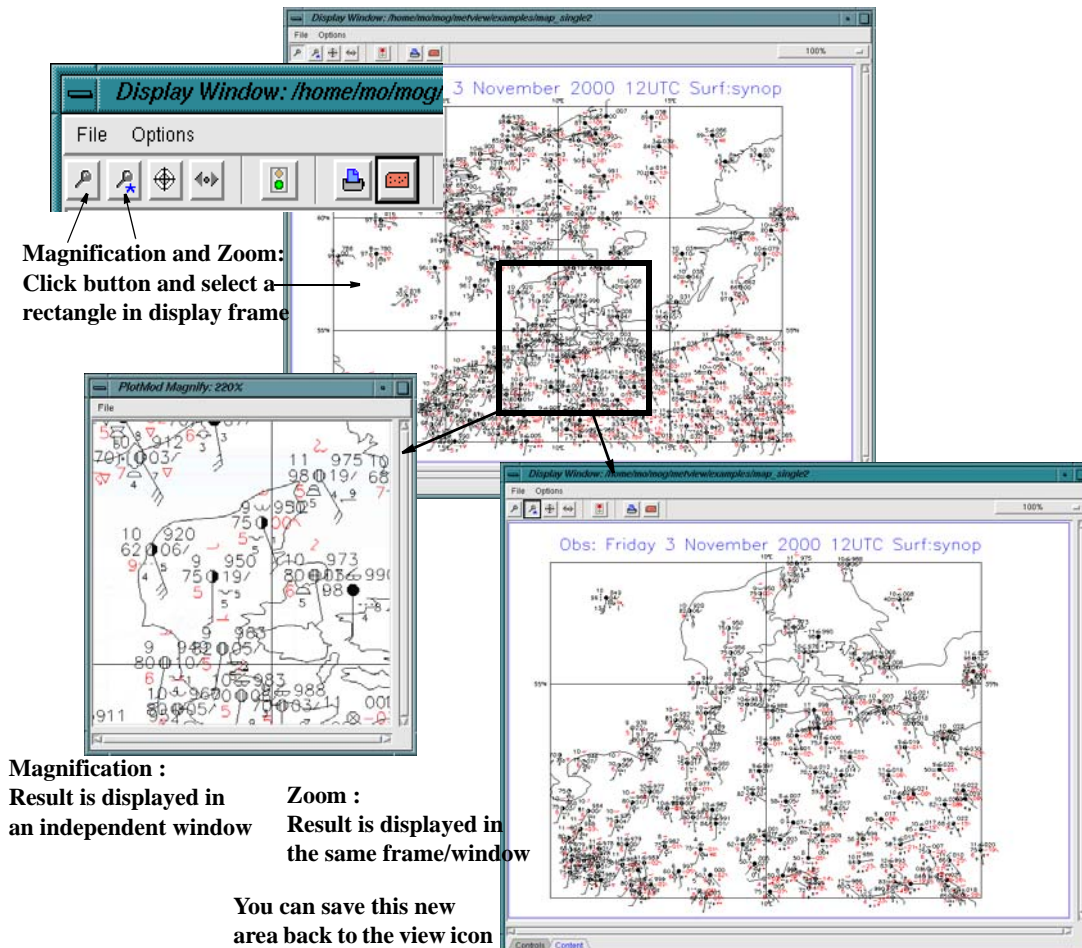


Figure I-59 : Magnification and Zoom on a visualisation of observation data. The magnification is a purely graphical enhancement displayed in a separate window, while the zoom redraws the visualisation within the same window

Changing the display geography

The display window allows you to change the geographic details (or *geography* for short) of a displayed plot, but this depends on the data unit being visualised :

- if you are visualising a multi-level, multi-time step set of model fields then you can change the geographic details at will, indeed you can even change the view from a Map view to a Cross-Section view (say).
- if you are visualising a NetCDF data unit of a single cross section along a particular transect, then you cannot change any geography detail, you are confined to the cross-section defined by the data unit and nothing else.

Geography modification is particularly useful in exploratory work usually from a direct visualisation; in this way, you can try several geographies before deciding on an ideal one.

When you've decided on a final geography you can then save the current geography to the view icon in use within your plot. If you're using a default view (e.g. you are doing a visualisation directly from the data icon), you will get a new view icon in your desktop. Saving the geography back to the view icon is detailed in the next section.

There are several ways to modify your geography :

- Using the Zoom facility - this is restricted to changing the area coordinates within a map plot; you can save the zoomed area to the view icon currently in use (see the previous section)
- Using the Change Geography option in the frame menu - this allows you to change all the geographical details in the view (subject to suitable data) and works by launching the **geography tool**. This is equivalent to the far more cumbersome method of going to the Contents drawer, editing the view icon and launching the geography tool from its help button
- Dropping a view icon - prepare a view icon with the required geography and drop it in the display window frame whose plot you want to modify. You can use this to change not only the geography but also the plot type by providing a view different from the current one (see "View Icon Drops" on page I- 120)

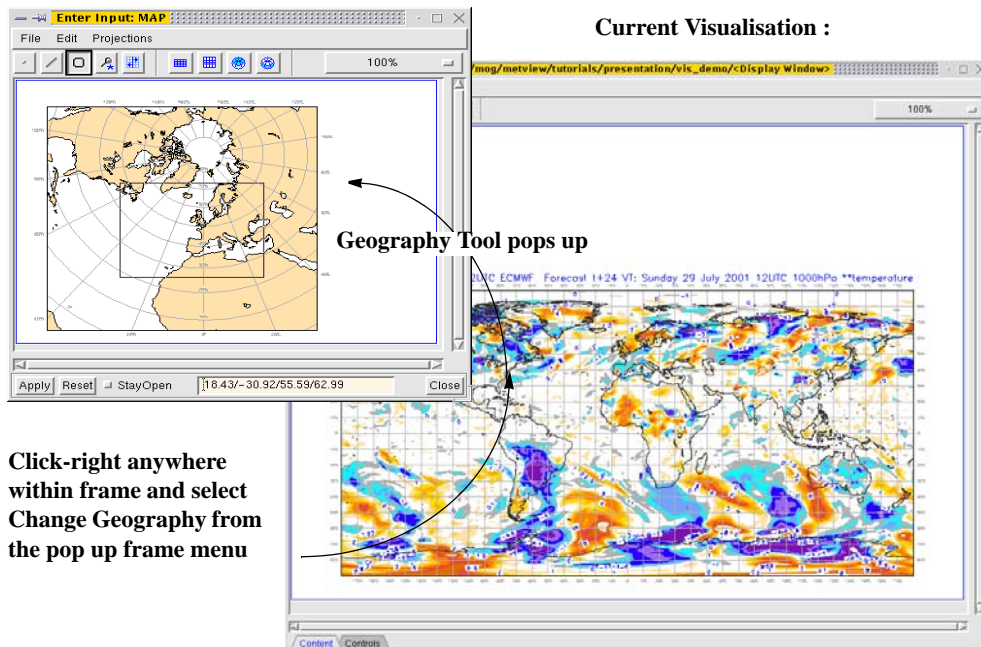
The **geography tool** is a graphical map-based interface which allows users to specify interactively geographical input such as coordinates of areas, lines or points as well as projection types. It has slight operative differences between those for area, line and point selection and is based heavily on the visualisation display window.

In this section we cover geography modification through the Change Geography option - so, to modify the geography of a plot :

- Click-right on the display frame to bring up the frame menu
- Select Change geography from the menu to launch the interactive geography tool
- Use the geography tool functionality to define a new region and/or projection

The process is illustrated in Figure I-60. For details of the geography tool functionality see "The Geography Tool" on page I- 91.

Geography Tool :
 Use its functionality to select
 a new region and/or projection



Click-right anywhere
 within frame and select
 Change Geography from
 the pop up frame menu

Figure I-60 : Changing the geography in a visualisation by means of the frame menu option

Saving the display geography

At any time in your visualisation work you can save the current geography displayed on screen. Note that *zoom actions are considered as geography modifications*. It is important to remember that all the modifications you make to the geography of a plot are held in memory. If you simply quit your visualisation, the geography modifications are lost.

However, you can save the geography modifications. Two situations may arise depending on whether you are using your own view icons or system provided ones :

- When using your own view icon the save action will save the geography modifications back to the view icon so as to make them permanent
- When using a system default view icon rather than one of your own, as in the case of a direct visualisation ("Direct Visualisation" on page I- 68), the save action will create a new view icon on your desktop

To save the geography modifications you can do one of the following :

- Select Save Geography from the frame menu - this is the most straightforward and convenient way
- Go to the Contents drawer, click-right on the view icon corresponding to the plot whose geography you modified and choose the **Save** option from the icon menu. You can do this even if the icon is the system default.

In the case of a system default icon, the save action will create an icon in the same folder where you launched the visualisation from. The icon will always have the name surrounded by angled brackets, e.g. <Map View>. Once created, you can use it as you see fit.

Obtaining values at a location

You can obtain point values from GRIB or BUFR format data visualised in a map view. The GRIB data *must* be retrieved as a lat-long grid. The process is exemplified in Figure I-61:

- Click-left on the point extraction button (see "Command buttons" on page I- 98) and the mouse cursor changes to a gun-sight when over the plot
- Click-left anywhere on the display window frame containing the data to be visualised. When you click the display, a pop-up window appears listing the data at the point coordinates

GRIB data - For GRIB data the output is simply the field value at the point location. The value is interpolated from the nearest grid-points. You only get a single value, the one for the visible plot, even when visualising a multiple data unit

BUFR data - For BUFR data, the output is the decoding of the BUFR message of the nearest location to the selected coordinates. The search area for the nearest data location is a square of 20 by 20 degrees centered on the selected location. Contrary to the GRIB data, all the data in the visualised BUFR data unit is decoded and printed - this means if you have several observation dates in the same display window, all the dates are decoded and listed, even though you are using only one of the plots for coordinate selection. If you have several observation types (e.g. synop and TEMP) at a location, all types get decoded and printed. You can choose to expand the BUFR bitmaps

GRIB and BUFR data - if you have both types of data in the visualisation, the point tool will output both. Note that in this case you can choose to have only one of the types displayed - select one of the options from the menu at the bottom of the output window

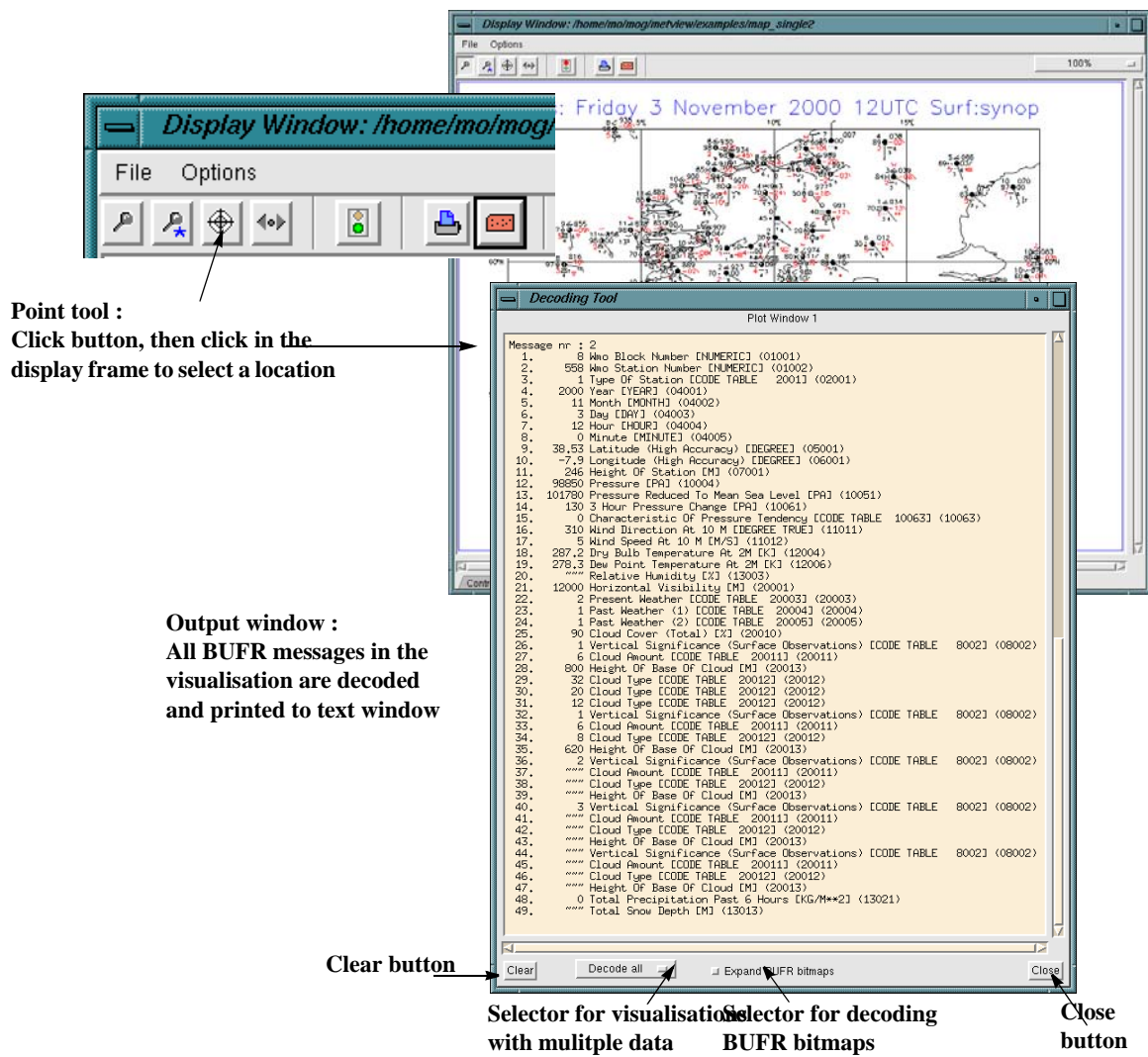


Figure I-61 : Using the point tool to obtain values from a visualisation of observations

Visualising stacks of plots

Overview

A data unit icon in Metview may contain any number of fields. When a multi-field data icon is visualised it leads to the generation of multiple plots - e.g. a multi-level data unit on a map view will generate one plot for each level, a multi-level, multi-forecast step data unit on a cross-section view will generate one plot for each forecast step. Sets of observations are split into six-hourly sets of plots - see "Observation Grouping" on page I- 119 for details.

A set of N plots in a display frame is known as a **stack of plots**. When you visualise a stack of plots, out of the N plots in the stack, a number M is visible at any one time, M being the number of connected frames you have - e.g. in Figure I-54, the bottom frame (3x1 connected) can display only 3 fields out of the total number contained in the data unit.

This is shown schematically for a plain single frame layout in Figure I-62 - the plots which aren't currently visible "sit" behind the visible ones waiting to be drawn. These undrawn plots are only

drawn the moment they are brought "to the surface", i.e. made visible on screen. Users make plots visible by scrolling through (or browsing) the stack of plots.

A 7 field data unit visualised on a single frame layout.

Initially, only the first field is visible, while the remaining ones sit on the background, waiting to be drawn.

To bring the background plots to the surface, you need to scroll through the stack of plots

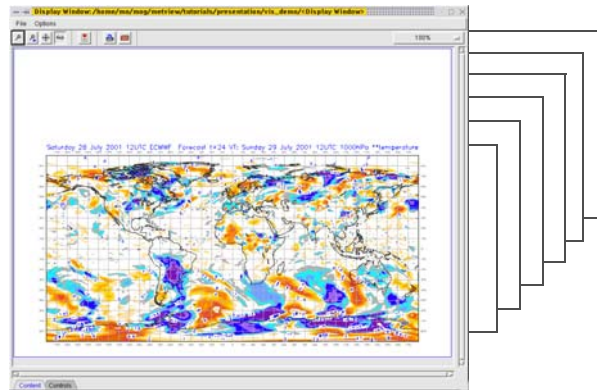


Figure I-62 : Illustration of the concept of a stack of plots. The display window provides all required functionality to scroll back and forth through the stack as well as browse its contents

Each time a plot is made visible it is drawn. Since drawing a plot on screen for the first time takes a finite amount of time, this would mean that visualising all 50 EPS members on a 1x1 plot (say), would require actively scrolling through all of them and waiting a couple of seconds at each and every step.

Metview allows users to override this default behaviour and make the display window draw all the plots (visible or not) in the background - this is known as *enabling background drawing*. When finished, displaying fields becomes nearly instantaneous since the plots are kept in memory - this is convenient if you have a stack with a long sequence of fields and would like them to be drawn automatically while you do something else. See "Background drawing of plots" on page I- 108.

Here we cover the ways in which you can scroll and browse through a stack of fields as well as enabling/disabling background drawing of fields.

Scrolling through a stack of plots

When you visualise a multi-field data unit, scrolling is turned ON automatically - the scroll toggle button appears pressed. You can turn scrolling OFF (or ON again) by clicking the scroll button as shown in Figure I-63. You have two ways to scroll back and forth a stack of plots :

Through the mouse cursor - this is the most convenient way, but requires the scroll toggle button to be ON. If scrolling is ON, you simply need to move the mouse over the plot and the cursor will change to an horizontal arrow. When the cursor is over the right half of the frame, the arrow points rightwards; clicking it moves *forward* the stack of plots by one plot at a time. When the cursor is over the left half of the frame, the arrow points leftwards; clicking it moves *backwards* the stack of plots by one plot at a time. If you have reached the end or beginning of the stack, the cursor will not convert to an horizontal arrow.

Through the frame menu - to scroll through the stack of plots, click right to bring up the frame menu and choose one of the options Scroll-> (to move forwards) or Scroll<- (to move backwards). These options are always available, irrespective of whether scrolling is ON or OFF. A further option is Scroll by screen. If this is ON (checked), when you scroll you will move by the number of connected frames (i.e. visible plots) in your display - on 2x1 connected frame layout, scrolling would replace

plots 1 and 2 of the stack by plots 3 and 4. If the option is OFF (default) scrolling would replace plots 1 and 2 with plots 2 and 3, as it advances plot by plot.

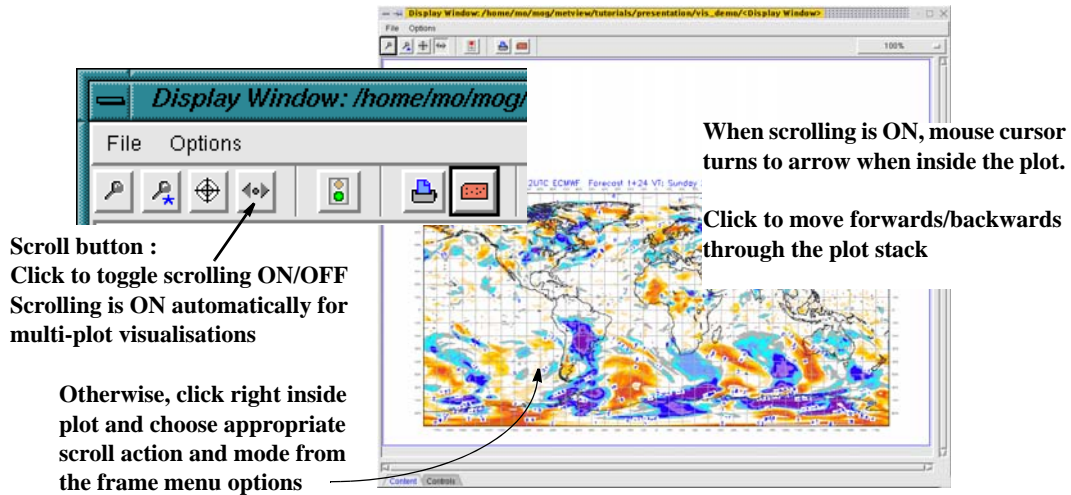


Figure I-63 : Using the scroll feature to move through a stack of plots on a single frame layout. See text for details

When you advance to a point of the stack of plots you haven't yet visualised, you will notice that the drawing of the newly uncovered plot hasn't yet been done, which means you may have to wait a few seconds for the plot to be fully formed. This happens because background plotting is disabled by default, i.e. Metview only draws those plots that are visible and not those which are sitting beneath them in the stack - you can control this feature as detailed in "Background drawing of plots" below.

Background drawing of plots

When you visualise a stack of plots, Metview by default will *only draw the plots that are visible*. A plot is only drawn when it comes into the screen for the first time (e.g. by scrolling forwards through the stack). This (default) behaviour is described as "*background drawing disabled*". **Background drawing** is simply the drawing of plots from a stack that haven't been yet displayed to the user.

Metview operates with background drawing disabled by default. However, Metview can also operate with "*background drawing enabled*". When background drawing is enabled, Metview will draw all the plots of the stack in the background - this may take a while depending on how many plots are generated and the resources of your machine.

What is the advantage of enabling background drawing? With background drawing disabled a plot is only drawn when it comes into view for the first time and this always takes a few seconds - once it is drawn you can visualise it much faster. If you have a very large number of plots in your stack, drawing all plots would require you to scroll through all of them, which would take a long while and be cumbersome. When background drawing is enabled, Metview draws all the plots in the background - this may take a while depending on how many plots there are in the stack and the resources of your machine. Once all the drawing is done, scrolling through the stack is virtually instantaneous.

The default behaviour is due to the fact that with background drawing disabled, you don't have to wait for the stack to be drawn and can proceed immediately with testing different visual definitions, modifying the display geography, etc.,. Once you are happy with the result, you can then scroll through the stack of plots, or enable background drawing.

To enable background drawing - click its toggle button (with a traffic light picture) on the command button bar as shown on Figure I-64. The traffic light icon will turn green and the drawing of the non-displayed plots will begin. Background drawing stays enabled and it will apply to subsequent data units dropped in the display window. Click its button again to disable it.

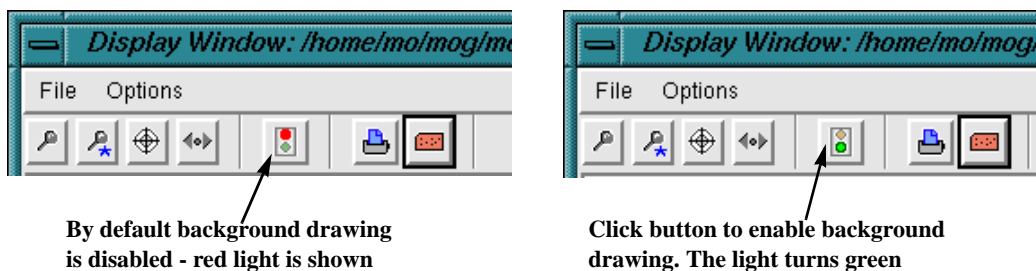


Figure I-64 : Enabling background drawing on a display window

The plot stack browser and progress meter in the Controls drawer allows you to check which plots have been drawn already - see "Browsing stacks of plots" below. If you have the Controls drawer open when you turn background drawing on, you will see the progress of the visualisation "live" as a green line streaming over an orange one.

Browsing stacks of plots

The display window has controls that allow you to handle a stack of plots arising from a visualisation. Given a visualisation with a stack of plots, you may need to :

- browse the stack of plots
- select one of the stack plots (without scrolling through all previous plots)
- check how many plots in the stack have been drawn

The functionality to carry out these tasks is contained in the Controls drawer which you open by clicking its tab. The controls include (see Figure I-65) :

- A **frame selector** - this selects the frame for which you want to control the stack of plots. The selector represents the frames composing the visualisation layout - click on a frame to select it. Shift click others for multiple selection.
- A **plot information** and **plot browser** - information on the currently displayed plot (parameter name, date, forecast step,...) is visible in a text field. Click the arrow button to its right to drop-down a list of all the data currently in the plot stack.
- A **plot slider** and **stack meter** - these allow you to display plots from the stack and to evaluate which plots have been drawn already.

This functionality applies to a single frame. If you select more than one frame, the plot browser becomes unavailable and the plot slider and stack meter refer to the frame with the lowest number (higher priority) - leftmost and uppermost.

To browse the plot stack, select one of the layout frames with the frame selector and click the browser button. The drop-down plot list shows you a summary of the data present in each display frame (plot). Click on the plot info to visualise the plot.

To select one of the plots for visualisation, select the plot out of the plot browser list; otherwise drag the plot slider until the plot appears in the display. If more than one frame is selected, you have to use the slider, which advances all plots simultaneously.

To check drawing progress of the plots in the stack, look at the stack meter. The plots already drawn appear as green, those to be drawn appear orange. If you have the Controls drawer open when you turn the background drawing on (see above) you will see the progress of the visualisation "live" as a green line streaming over an orange one.

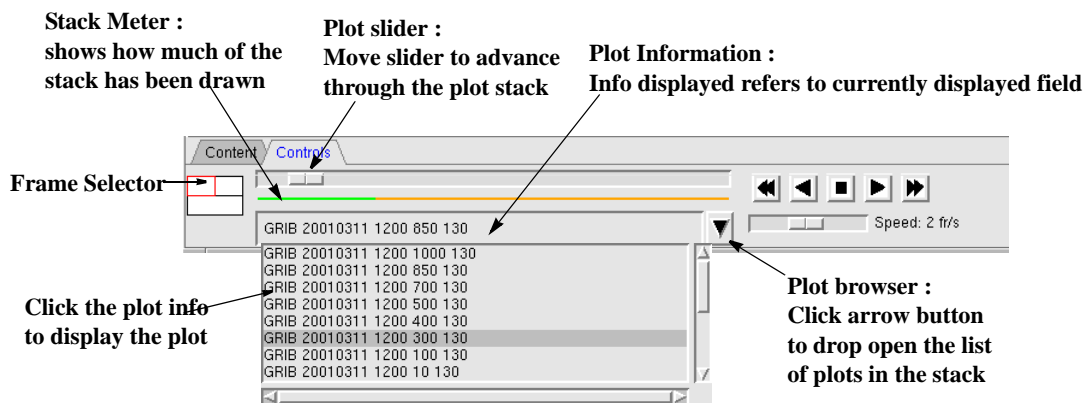


Figure I-65 : The controls for browsing through the visualisation. You can gauge the progress of the drawing, browse the plot stack and pick a plot out of the stack

Animating stacks of plots

You can animate any sequence of plots whether to gain some insight or simply to visualise/browse a long series of plots with little effort. The animation functionality is available in the Controls drawer as shown in Figure I-66 and allows users to :

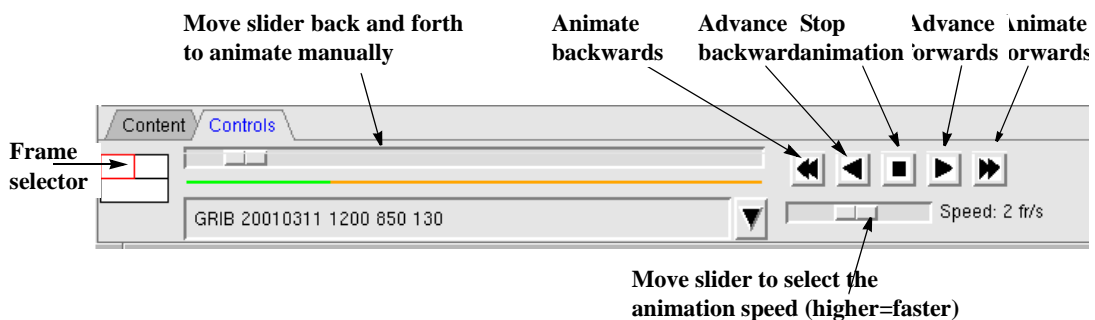


Figure I-66 : The animation controls. Users can animate the plot stack or move through the stack one plot at a time. Slider movement allows for manual animation at arbitrary speed.

Animate the stack of plots - you can animate back and forth clicking the animation buttons. The animation speed is controlled by a slider which specifies the number of plots per second, hence the higher the number, the faster the animation. You can also animate the stack of plots manually by

moving the plot slider back and forth. This allows animation at arbitrary speed. You can stop the animation by clicking the stop button.

Advance one plot at a time - you can move back and forth the plot stack by clicking the advance buttons.

Synchronised animation - When you have several frames in the visualisation you can animate them in synchrony. This synchronisation is based on the order of the plots in the stack, not on the time stamps, levels or ensemble numbers of the displayed data. To animate multiple frames in synchrony, simply select all the frames with the frame selector (drag over or shift-click all frames) and click the animate button.

Synchronised animation starts from the currently displayed plots and you can use this to animate plots of related variables shifted by any number of plots, e.g. animate fields in different frames where data in one of the frames is separated from the data in the other frame by a number of forecast steps (say).

No checks are done on the number of plots in the different frames. If they contain different numbers of plots, they will get in and out of sync in a pattern dependent on the relative number of plots in each frame.

Converting a visualisation to a macro program

This facility allows users to save a simple visualisation as a macro program. The generated macro will faithfully reproduce the display window layout, visual definitions and basic data retrievals. For more complicated data you may have to edit the macro manually to ensure correctness. The macro program is saved to the directory you are working from, and is obtained by clicking-left on the Save as Macro button (see Figure I-67). The resulting macro is always named Plot page contents N, where N is the number of files with this name already present.



**Click on Save As Macro button.
A macro which will regenerate the current visualisation**

Figure I-67 : Saving the visualisation as a macro program

You may use this facility to :

- save yourself the work of rebuilding the visualisation from scratch and retaining all the icons involved in it
- obtain a macro program which you can edit and alter in order to produce other visualisations or use as part of other programs

To recover the visualisation, *execute* or *visualise* the macro icon. You can also modify it or rename at will.

Printing a visualisation

You can print your on-screen visualisation to a variety of outputs :

- Printer
- PS file on disk
- PS viewer (ghostview/showps)
- JPEG or PNG graphics file

The display window printing facility is available from the print button. Click the Print button (see Figure I-68) to launch the Print dialogue editor (shown in Figure I-69).



Click on Print button to launch Print dialogue

Figure I-68 : Launching the Print dialogue

In the print dialogue editor you enter the required printing options and then click Apply to carry out the printing. The specifications you enter here remain from one call to the Print dialogue to another.



Figure I-69 : The print dialogue editor. The editor is the same for setting the printer preferences or carrying out the actual printing. Both take place on clicking the Apply button

The print dialogue editor has the following components :

Format - Defines the output format : Postscript, JPEG, PNG or Screen

Destination - Defines the output destination : this can be Preview, File or Printer. If using Preview you need to specify Preview Program (see below). A choice consistent with the output format will be made automatically by Metview in case of incompatibility (previewing a JPEG with ghostview)

Preview Program - Select a preview program for what was specified in Format.

Print Option - Choose whether to print only the displayed plots (Visible) or the full plot stack (All).

Ncopies - Enter the number of print-out copies. Note that this option only works if **Destination** is set to Printer.

File Name - Enter a file name if printing to a file (needs destination set to File). If no path is specified the file is written to the current directory

Width in Pixels - Enter this quantity if producing a JPG or PNG graphics file

Jpeg Quality Level - Enter if producing a JPEG file. Number should be from 0 to 95 (increasing quality and size) or -1 to obtain the default JPEG quality level (usually a reasonable compromise between quality and size)

Printer - Pick a printer from the list of available ones. When printing to file, the printer chosen specifies the output characteristics.

Printer Customize - Toggle Yes/No for printer customization.

Printer Type - Enter the printer type. Needs Printer Customize set to Yes.

Printer Resolution - Enter the printer resolution. Needs Printer Customize set to Yes.

Printer Scale Factor - Enter the printer scale factor. Needs Printer Customize set to Yes.

Printer Path Size - Enter the path size. This is the maximum number of points that a printer may use in order to plot an individual line segment. Internally, lines may be made up of more than one segment if there are more points in the line than the path size permits, making this parameter purely a matter of efficiency. Default should be correct for each printer. Needs Printer Customize set to Yes.

Printer Stack Size - Enter the printer stack size. Needs Printer Customize set to Yes.

Printer Miter - Toggle the printer miter setting. Needs Printer Customize set to Yes.

Print Command - Enter the specific print command

Icon Definition File - Determines whether a dot file (see "Icons Revealed" on page I- 32) is generated for the resulting file (if **Destination** is set to File). As the dot file may be misinterpreted by some external programs, it may be desirable to prevent its creation. Note that Metview will automatically create a dot file if the directory containing the output file is subsequently opened from within Metview.

Once you specify the print settings, click Apply. to carry out the printing. If you print to a file and store it within the Metview environment, it will be assigned a specific icon (PS, JPEG or PNG). To print the file, *Visualise* it first - this will launch ghostview, showps or xv, depending on the file and system specifications - and print from the viewer program.

Note that the plotting behaviour is slightly different between the various file formats. With PostScript, each subframe is plotted on the same page within a single output file. However, because JPEG and PNG do not support multiple pages and are designed primarily for compressed, medium quality web output, each subframe is plotted to a new file. The next section describes the filenaming conventions for JPEG and PNG plots.

Filenaming Conventions for JPEG and PNG Plots

Because the JPEG and PNG file formats do not support multiple pages, Metview will instead create multiple files - one for each page of the plot. The behaviour under JPEG and PNG output is slightly different, and best illustrated by example.

For JPEG output where the filename has been specified as plot.jpg, a two-page plot will produce the following two files:

```
plot.jpg.1.jpeg  
plot.jpg.2.jpeg
```

Thus it perhaps makes sense to omit the extension from the filename specification. Giving a base filename of plot, the resultant files will be:

```
plot.1.jpg  
plot.2.jpg
```

For PNG output where the filename has been specified as plot.png, a two-page plot will produce the following two files:

```
plot.1.png  
plot.2.png
```

In this case, it makes no difference whether you specify the filename as plot.png or plot.

Note that in both cases, the automatic addition of numbers to the filenames is performed even if the plot has just a single page.

Visualisation Input by Icon Drag and Drops

The Metview way to provide input (data, visual definitions) to a display window on screen is by **icon** (drag and) **drop** (see example in Figure I-43 on page I-69).

To **icon drop**, simply drag an icon from the desktop and drop it inside (one of the frames of) the display window. This a single icon drop. You can also drop several icons at the same time (**joint icon drop**) by selecting a number of icons (see "Icon Selection and Icon Menus" on page I- 21) and dragging them all inside the display window frame.

You can drop data icons, visual definition and view icons. The dropping of icons allows you to add data to the visualisation, replace or add to the visual definitions in use and modify or replace the current view :

- **Data icons** - when you drop a data icon, the data is retrieved (e.g. MARS Retrieval), computed (e.g. macro returning data) or read (e.g. data file in disk) and then plotted with available visual definitions (either one provided by the user or some default one). The data icons you drop must be consistent with the view currently in the frame. When you drop several data icons (whether sequentially or jointly) in a display window, which data is displayed first depends on how quickly it is read from disk / returned from the database / computed by the application or macro. If the order of return is relevant for your purposes you must wait for each data in turn to be displayed, before dropping the following data icon. See "Data Icon Drops" on page I- 116.

- **Visual Definition icons** - when you drop a visual definition, Metview applies it to the plotting of the data currently available on the display frame. Note that several visual definitions jointly dropped behave as a single visual definition. See "Visual Definition Icon Drops" on page I- 119.
- **View icons** - when you drop a view icon in a display window frame, the current view is replaced by the newly dropped view. Recalculation takes place, e.g. to generate a new plot consistent with the new view. See "View Icon Drops" on page I- 120

In practice, many icon drop scenarios involving data and visual definition icons can be envisaged; for example :

- You can provide multiple data to the same visualisation, e.g. several parameters on several levels and/or forecast steps and/or dates and/or ensemble members. You can provide this as a single data icon (GRIB File or MARS Retrieval) or as a number of data icons, which you can drop together or sequentially.
- You can provide several visual definition icons to the same plot and these can be dropped jointly or sequentially; you may have a single data unit in the plot or you may have several.
- You can drop visual definition icons simultaneously with data icons, or you can drop a data icon followed by a visual definition icon, or you can drop all data icons and then all visual definition icons.
- Dropped icons can be new to the visualisation or may have been already dropped before, e.g. you have modified a detail in the icon and now repeat its drop.

So we see that icon drops may require complex rules to guarantee that users obtain sensible and consistent output from the large variety of icon drops that can be envisaged. Three problems stand out :

Consistency of Drops - You can drop icons which are incompatible or that do not lead to meaningful visualisations, e.g. wind arrows on a scalar field or single level data unit into a cross-section view. This is handled by simple checks on the dropped and resident elements.

Data Overlay - For both operational and research uses it is frequently required to produce plots with more than one data parameter overlaid, e.g. temperature and geopotential. If you visualise multiple data units you need to know what data combinations and drop types lead to overlaying of data and how this overlaying can be controlled.

Visual Definition Assignment - A single plot may have overlaid data units each requiring its own visual definition. Or a single data unit may require two visual definitions. If you provide new visual definitions what happens to the existing ones? You need to know which icon drops lead to the correct assignment of visual definitions to data units.

Metview has rules for the handling of icon drops with respect to the problems above :

- Data icon drops obey the *data overlay rules* which control the overlaying of different data units in the same plot
- Visual definition icon drops obey *visual definition assignment rules* which control which visual definition goes with which data unit

These are detailed in the next sections.

Data Icon Drops

Dropping data icons in a display window frame leads to the plotting of the data. This is simple enough, but when visualising multiple data icons involving several parameters, forecast steps, levels, etc, in the same display window frame, you need some rules to define which (if any) data is plotted together in the same plot.

Data overlay rules and overlay controls

Data overlay rules are the rules which determine how the multiple data is to be plotted, i.e. they determine if and how Metview overlays different data in the same plot and when it places them in different plots.

Rule 1 - Multiple data contained in a single icon or file (data unit) are never overlaid

Example : If you drop a single MARS Retrieval icon retrieving n Z fields and n T fields, this results in $2*n$ plots in the display window; the sequence of plots is [Z1, T1, Z2, T2, ..., Zn, Tn] if you specified "Z/T" in the MARS Request editor, or [T1, Z1, T2, Z2, ..., Tn, Zn] if you specified "T/Z".

Rule 2 - Multiple data provided in several data icons is overlaid according to user or system defined **overlay controls**. The overlay controls allow the user to :

- Never overlay the different data units
- Always overlay the different data units
- Control the data overlaying by means of **overlay settings**

Overlay settings work in the following way :

- The user selects one or more attributes of the data (e.g. forecast verification date, model or pressure level) to act as overlay settings.
- Only the data units with the same value for the selected attributes are overlaid; otherwise they are plotted separately.

The data attributes that can be used as overlay controls are :

- verification date
- verification time
- pressure or model level
- areal coverage of data (to be used with the map view only)

The overlay controls (never / always / use settings / attributes to use as settings) are a property of the View icon (see "Data overlay control in view icons" on page I- 88) used in the display window frame where the icons are dropped and are specified by means of an embedded Overlay Control icon.

In the Overlay Control icon you can select the Overlay Mode (never / always / check settings) and specify which data attributes to use as overlay settings by means of a simple list of ON/OFF toggle switches.

When a data attribute is selected as an overlay setting, all the data fields for which the attribute value is identical, are plotted together (or overlaid). The more attributes you select as overlay settings the more restrictive data overlay becomes.

Data overlay in practice

When you have several data icons to be plotted in a display window, the actual process is as follows (assuming you have your display window up and running) :

- A first data icon is dropped and its contents plotted leading to a stack of n plots
- A second data icon is dropped. Its data (e.g. fields) are examined in sequence and compared to those already in the display window, considering the overlay controls specified in the view :

If the overlay mode is set to "never overlay", the fields are simply plotted on their own at the bottom of the stack

If the overlay mode is set to "always overlay", the fields are plotted over the existing ones in the order they arrive

If the overlay mode is set to "use overlay settings", the data attributes selected as overlay settings are compared to those of the residing data - if the values of these attributes are the same the data fields are overplotted; otherwise the data field is sent to the bottom of the display stack

- Other data icons which are dropped follow the same previous rule

This indicates that you can get a completely different overlay of data depending on which overlay settings are set and also depending on the order in which you drop the icons.

Note : The plots arising from the *first* icon never change their position in the stack and will always be in some well defined order (e.g. of model level, forecast step). An example makes this clear :

Suppose you have two data icons, each with a number of forecast model fields at a variety of forecast time steps - one icon of geopotential fields (Z) at 500 hPa, with forecast steps of 24, 48, 60, 84 and 96 hours; a second icon of temperature fields (T) at 850 hPa, with forecast steps of 24, 36, 48, 72, 96 hours. The table below indicates the verification dates and times relative to an arbitrary reference date D for the two icons. As you can see some Z and T fields have the same verification date and time, others only the same verification date while others only the same verification time.

Z icon	D 12 500 hPa	Fc Step	Z24	Z48	Z60	Z84	Z96
		Verif Date	D+1	D+2	D+3	D+4	D+4
		Verif Time	12	12	00	00	12
T icon	D 12 850 hPa	Fc Step	T24	T36	T48	T72	T96
		Verif Date	D+1	D+2	D+2	D+3	D+4
		Verif Time	12	00	12	12	12

We now detail what happens when you visualise these two icons in a Map View under different combinations of overlay controls :

OVERLAY MODE = ALWAYS OVERLAY

When set to "always overlay " all data fields are overlaid irrespective of their attributes and hence the result is (data in brackets are overlaid) :

(Z24,T24) / (Z48,T36) / (Z60,T48) / (Z84,T72) / (Z96,T96)

OVERLAY MODE = NEVER OVERLAY

When set to "never overlay " all data fields are plotted separately irrespective of their attributes and hence the result is one of :

Z24 / Z48 / Z60 / Z84 / Z96 / T24 / T36 / T48 / T72 / T96
T24 / T36 / T48 / T72 / T96 / Z24 / Z48 / Z60 / Z84 / Z96

depending on which data icon is dropped first

OVERLAY MODE = CHECK SETTINGS / OVERLAY DATE = ON / OVERLAY TIME = ON

When set to "check settings ", and specifying the (verification) date and time as overlay settings, the result is (data in brackets are overlaid) :

(Z24,T24) / (Z48,T48) / Z60 / Z84 / (Z96,T96) / T36 / T72
(T24,Z24) / T36 / (T48,Z48) / T72 / (T96,Z96) / Z60 / Z84

The first (second) arrangement is obtained if you drop the Z (T) icon first. Note how the first dropped data is arranged in order of forecast step; the second dropped data is then arranged in such a way as to obey the overlay settings. Hence the order of the drop leads to a different sequence of (identical) plots. Unmatched data from the second icon is placed at the end of the stack.

OVERLAY MODE = CHECK SETTINGS / OVERLAY DATE = ON / OVERLAY TIME = OFF

When set to "check settings ", and specifying the (verification) date (but not the time) as overlay settings, the result is (data in brackets are overlaid) :

(Z24,T24) / (Z48,T36) / (Z60,T72) / (Z84,T96) / Z96 / T48
(T24,Z24) / (T36,Z48) / T48 / (T72,Z60) / (T96,Z84) / Z96

The first (second) arrangement is obtained if you drop the Z (T) icon first. Note how the first dropped data is arranged in order of forecast step; the second dropped data is then arranged in such a way as to obey the overlay settings. Hence the order of the drop leads to a different sequence of (identical) plots. Unmatched data from the second icon is placed at the end of the stack.

OVERLAY MODE = CHECK SETTINGS / OVERLAY DATE = OFF / OVERLAY TIME = ON

When set to "check settings ", and specifying the (verification) time (but not the date) as overlay settings, the result is (data in brackets are overlaid) :

(Z24,T24) / (Z48,T48) / (Z60,T36) / Z84 / (Z96,T72) / T96
(T24,Z24) / (T36,Z60) / (T48,Z48) / (T72,Z96) / T96 / Z84

The first (second) arrangement is obtained if you drop the Z (T) icon first. Note how the first dropped data is arranged in order of forecast step; the second dropped data is then arranged in such a way as to obey the overlay settings. Hence the order of the drop leads to a different sequence of (identical) plots. Unmatched data from the second icon is placed at the end of the stack.

In this simple example the other overlay settings lead to fairly obvious outcomes. If you set OVERLAY LEVEL=ON, no overplotting would take place since the data do not share the same pressure level. If they covered different areas and you had set OVERLAY AREA=ON, again no overplotting would take place.

Plotting Geopoints Over Fieldsets

The rules for overlaying geopoints data with fieldset data are as follows:

- The date and time of the first geopoint are checked against each field in the fieldset. Note that the time should be expressed in HHMM format; a time value of 12 will be interpreted as 0012, i.e. 00:12.
- If there is a match, then the geopoints are overlaid with the first matching fieldset.
- If there is no match, then the geopoints are plotted by themselves on a new page
- If the date and time of the first geopoint is missing or invalid, the geopoints are overlaid with the first field.

Observation Grouping

When an icon returning observation data is dropped onto a display window, the observations are, by default, grouped into 6-hourly sets with each set occupying its own plot. There are two parameters in the Overlay icon that modify this behaviour. First, it is possible to change the number of hours in the sets (default 6). Second, it is possible to specify how many minutes into the hour an observation group should start. By default, an offset of one minute is used. This leads, for example, to a group ranging from 03:01 to 09:00. To obtain a group ranging from 03:00 to 08:59, use an offset of zero minutes. It may be worth checking the output messages (see "The Message Window" on page I-126) when you visualise observation data because the time ranges for all the groups created will be printed for your information; you can use this to help you create a more precise grouping. The observation grouping parameters are available in the Overlay icon - see "Overlay Control" on page III-237.

The default groupings are 03:01 - 09:00 UTC, 09:01 - 15:00, 15:01 - 21:00 and 21:01 - 03:00.

It is worth noting that these two observation grouping parameters are set globally when the owning view is visualised. This means that if two views with different grouping parameters are both visualised before any observation data is dropped onto them, then the grouping parameters attached to the second view to be visualised will actually be applied to both views. The way to use different grouping parameters for different views is to visualise a view and then drop data onto it; then do the same with the second view.

See "Obtaining values at a location" on page I-105 for a caveat regarding the operation of the Point Extraction tool on observations that have been split into groups.

Visual Definition Icon Drops

Dropping a visual definition icon in a display window frame with a single data unit leads to a simple outcome of the data being plotted with the dropped visual definition. However, if you have a number of different data units and you need different visual definitions to be applied, you need some rules to assign the right visual definition to the right data unit.

Visual definition assignment rules assign visual definition icons to the data units composing the visualisation. These rules define the outcome of the many ways in which users can provide data and visual definitions to a display window, e.g. :

User adds a new visual definitions to a data unit already displayed

User drops data icon and visual definitions in sequence

User joint drops data icon and visual definitions

User joint drops adds two visual definitions (to be applied to the same data icon)

Note that rules for these icon drops can only resolve some of the ambiguity of outcomes, but can't provide the full flexibility users may require. The reason is that you cannot assign explicitly a given visual definition icon to a particular data unit present in the visualisation simply by dropping its icon in a display window. If you want to attach a visual definition to a particular data unit, you may have to go through the Contents display which gives full and explicit control over visual definition drops - see "Icon Drops in Contents" on page I- 124.

The rules for visual definition matching are as follows :

1 - A visual definition dropped in a display window frame is assigned to *all* data units present in that frame which can accept that visual definition. Existing visual definitions are removed by the new one. If the visual definition is not consistent with the data (wind arrow on scalar field) it is not applied.

2 - If a visual definition icon is already present in the frame, dropping it again (e.g. after modifying some of its details) simply replaces its previous instance, leaving the remaining plot components untouched.

3 - A visual definition dropped *jointly* with a data unit remains assigned to this data unit and will not be applied to any other already present in the display.

4 - Two visual definitions *jointly* dropped act as a single visual definition in that both will apply to the same data unit; this is common when visualising difference fields using a visual definition for the negative values and another for positive values.

These rules determine the outcomes of visual definition icon drops on display windows. It is clear that the sequence of icon drops and whether these are of single or joint defines your end result.

If you need to have full control over which visual definition icon is assigned to which data unit without the constraints of the assignment rules, you have to use the Contents drawer - this provides an interactive representation of the visualisation structure where you are free to assign visual definitions to data explicitly as well as modifying or removing them - see "Assignment of visual definitions" in "Using the Contents Window" on page I- 123.

View Icon Drops

Dropping a view icon in a display window frame leads to the replacement of the existing view. The outcome of such an action depends on :

- whether you are dropping a view icon of the same type
- whether the existing data is compatible with the new/revised view

If you drop a view of the same type, obviously the plot perspective will remain the same, but you can provide :

- a new region and/or geographic projection (Map View)
- a new transect line and axis details (Cross Section View)

- a new averaging area and axis details (Average View)
- a new point/area and axis details (Vertical Profile View)
- new plot dimensions and frame details (all views)

The data you are visualising must be compatible with changes to the view. If your data is composed of global model fields, then any change to the view can be accommodated. However, if you are visualising a partial area of the globe, your new modified geography may lead to partial or incomplete plots. If you are visualising a cross-section NetCDF file, you cannot change any geographic detail of the cross section, though you can change some details of the axis or plot dimensions.

If you drop a different view, then a new plotting perspective is applied and you get a new type of plot. You can use this to turn a map plot into a cross-section (say).

Again the data must be compatible with the changes of view. You must be visualising multi-level model fields if you want to produce a cross section, average or vertical profile from a map visualisation. If you are visualising finished data (cross section NetCDF) the change of view is ignored.

Using the Contents Window

Overview

A visualisation on a display window is defined by its component icons - the layout and views, the data and visual definitions, and by the way they are structured (how many frames there are in the layout, which views and data go into what frame, which visual definitions are assigned to which data unit).

The display window provides access to the component icons of the visualisation through the **Contents drawer**. An example is presented in Figure I-70 - the Contents drawer is basically an *interactive representation of the visualisation structure* and it :

- provides the user with a schematic display of the visualisation structure showing all the icons involved, their relative role and their positioning within the visualisation layout
- allows the user to modify interactively the icons that make up the visualisation, add new icons to the visualisation and replace or remove existing ones

In a word, the Contents drawer offers users a very fine degree of control over the visualisation process. Indeed some modifications to a visualisation may only be possible to do via the contents drawer.

The contents structure

Content Levels - the icons composing the visualisation are structured in three levels and each level can have one or more divisions (see Figure I-70) :

- **Display Level** - the topmost level representing the display window with all its frames. By definition, this level can only have a single division. Only the Display Window icon and Visual Definition icons can reside here.

- **View Level** - the level of the individual frame. This level has as many divisions as there are frames in the layout. Only a single view icon and visual definition icons can reside in each division.
- **Data Level** - the level of the individual data icons. This is conceptually the lowest level. There are as many divisions as data units in all the display frames. Only a single data icon and visual definition icons can reside in each division. The divisions are grouped by the view level division their data icons belong to.

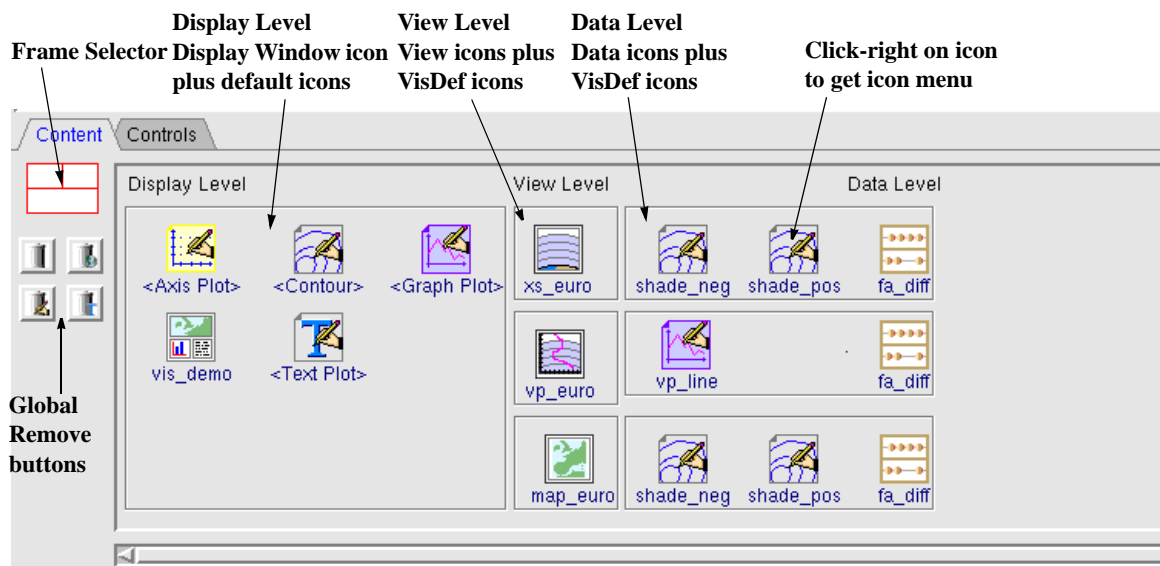


Figure I-70 : Opened Contents drawer for the sample visualisation in Figure I-54, showing the three different levels of a visualisation structure. Each icon can be individually manipulated and you can add icons to the contents, allowing fine control of the visualisation

The Contents structure is hierarchical, in that a level contains and encloses subordinate levels. The level at the top of the hierarchy sits at the left of the window, whilst the bottom level sits at the right. Icons residing in a given level will interact with all the icons of all the lower levels - this is known as the *icon cascade* and is relevant to understanding the outcomes of icon drops in the contents structure (see "Icon Drops in Contents" on page I- 124). Icons residing in separate divisions of the same level are fully independent from one another.

The icons in the Contents drawer have the same name as their desktop instances. However, it frequently happens that a visualisation employs system default icons for those parameters that users do not specify (e.g. one for the plot title). System default icons appear in a Contents drawer with names within angled brackets (<>) e.g. <Text Plot>. They are always placed in the Display Level, except for the default view icons (used in direct visualisations) which go to the View Level.

Frame Selector - on the top left corner of the Contents window you can find a frame selector. This widget replicates the plot frame arrangement of your visualisation. By clicking one of the frames only the corresponding plot is selected and only the divisions corresponding to this frame are shown in the contents. You can select more than one frame by a click drag across both frames or by clicking the first frame and shift-clicking the other frames to be selected.

Icon Remove Tool - below the frame selector you can find a set of four buttons; these allow you to remove icons both all of them or all of a given type (visual definition or view) from the currently selected view (see "Modifying a visualisation contents" below).

Assignment of visual definitions

The display window icon, view icons and data icons define their own Contents level and always reside in there. However, the visual definition icons can reside in any of the three levels. Which data icon(s) they are assigned to depends on the type and relative position of the other visual definition icons available in the visualisation. The rule is :

- A data icon uses the compatible visual definition icon which is closest to it

The assignment process is a simple search by the data icon for the nearest compatible visual definition icon :

- first it looks in its own Data Level division
- if nothing is found, it looks in the next level up, the View Level it belongs to
- if none is found, it looks in the next (and final) level above, the Display Level

This implies that a visual definition icon which resides in a given level applies to *all* data icons of the levels below, provided :

- it is compatible with them
- and no other visual definition takes precedence (i.e. by residing in a lower level).

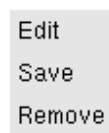
This is why default visual definition icons are placed in the Display Level - staying at the top level they are guaranteed to be found by all the data icons of lower levels which have no other visual definition to use. Default visual definition icons cannot be removed if they are in use; if they have been removed and they are required again, e.g. if you remove all the visual definitions below them, they will be re-created automatically.

Modifying a visualisation contents

The contents drawer provides functionality to enable you to manipulate (some of) the visualisation component icons. Any modification in the component icons is immediately applied to the visualisation. You can :

- remove data and visual definition icons
- edit view icons
- save modifications of edited icons back to their desktop instances
- add (drop) new icons

This functionality is provided by means of an icon menu (to remove, edit and save icons) and icon drag and drops (to add new icons). The icon menu is available from a click-right on the icon picture :



Editing icons - Select Edit from the icon menu. The icon editor is launched. Modify at will and Apply/Close. *Only visual definition icons and view icons can be edited.* The modifications you introduce are applied to the visualisation but are not saved back to the desktop icon, until you do it explicitly by means of the Save option described immediately below.

Saving icons - Select **Save** from the icon menu. This saves any modification you might have done to an icon, back to its desktop instance. If you do this to a default icon (name enclosed within angled brackets), you will create a new icon with the same name. You can use this feature to interactively create new (view or visual definition) desktop icons from modified default ones - this has the advantage that you can see the effect of the changes on the visualisation as you make them.

Removing icons - Select **Remove** from the icon menu. The icon is removed and the visualisation is redrawn without it. Only visual definition icons and data icons can be removed. Default visual definition icons can be removed, but are instantly re-created if a data icon is left without a visual definition assigned to it (see "Assignment of visual definitions" above).

When you remove data icons, the division where the icon resides is also removed with all its contents, i.e. any visual definition icons that might have been associated with the data icon.

When you remove a visual definition icon, the data icon(s) using it will look for a new one, following the rule outlined above :

- *A data icon uses the suitable visual definition icon which is closest to it*

Contents also provides a global remove facility. You can :

- remove all data and visual definitions
- remove all data icons
- remove all visual definitions
- remove all text related (text and annotation) icons

this removal action is applied to the views that are *currently selected*.

Adding new icons - New icons can be added by dropping them in. You can drop new icons in whichever level of the contents drawer you want. The results differ depending on which level you drop the icon in and the type of icon it is. See next section for details.

Icon Drops in Contents

In a Contents structure you can only drop View icons, Data icons and Visual Definition icons. When you drop an icon, the visualisation is redrawn taking the new icon in consideration. The drop outcomes and behaviour are as follows :

Visual Definition icon - since these icons can reside in every level of the contents structure, *they will stay exactly in the level they were dropped in*. When the visualisation is re-drawn, the visual definition assignment rules are replayed taking the dropped visual definition into account - see "Visual Definition Icon Drops" on page I- 119;

Note that contrary to what happens when you drop visual definitions in a display window frame, no automatic removal of other visual definitions takes place at all. Visual definitions dropped in the Contents stay in the level they were dropped in and add their effect to the existing ones. If you need to remove other visual definitions you must do so explicitly, by means of the **Remove** option in the icon menu.

Data icon - since this icon must reside in the Data Level, it ends being placed here, irrespective of which level you dropped it in. If you drop it in a View Level it will create a new division for itself in that view's Data Level. If you drop it in the Display Level it is passed on to all the available views in

the View Level. From here it will create a new division in each of the view's Data Level. In a word, data dropped in the Display Level migrates down to all Data Level of all the views. When the visualisation is re-drawn, the new data is incorporated into the plots and the exact outcome will depend on the rules of data overlay specified in each view (see "Data overlay rules and overlay controls" on page I- 116).

View icon - since these icons must reside in the View Level, it is here that they end up being placed, irrespective of where you drop them. If you drop it in the View Level, the existing view is replaced by the new one; the same happens if you drop it in a Data Level division of the residing view. If you drop it in a Display Level, it replaces the view icons in all the view levels below. When the visualisation is re-drawn, the new views are used - depending on what you had and which view you drop you may get a completely new plot type.

METVIEW UTILITIES

This chapter describes and details a set of Metview Utilities. These are available to users from the menu bar of the main User Interface. They comprise the Message window (**File/Messages**), the Help system (**Help** menu) and the Metview tools (**Tools** menu).

The Message Window

Metview in the course of its operations generates a number of system messages and other text output, including :

- Error messages, e.g. from a failed data retrieval or any error leading to a *red icon* status (see "Icon Feedback" on page I- 26)
- Information messages , e.g. data retrieval generated information. E.g. a data retrieval will generate a message like :

```
[RH] Requesting 9 fields
[RH] INTFB: Resolution automatically set to 106
[RH] 9 fields retrieved from 'vpp5000 FDB'
[RH] 9 fields have been interpolated on 'lysander'
[RH] Request time: wall: 3 sec cpu: 2 sec
[RH] Visiting vpp5000 FDB: wall: 3 sec
[RH] Post-processing: wall: 2 sec cpu: 2 sec
```

- Text output generated by an icon - macro programs' console output (from function print())

All of this text output is directed to and displayed in a Message Window. Within a Metview session, this Message window opens up automatically the first time you carry out an operation that generates a message. You can close the window but all messages will still be directed there. You can open the window again by selecting option **Messages** from the **File** menu.

Text from the message window can be copied and pasted to text files - this may be useful information if you are requesting troubleshooting assistance. If you need you can erase the currently stored messages by clicking the **Clear** button on the message window.

If you are generating a large number of messages, say from a macro program with a print() command inside a for loop, which may be difficult to read, you can stop them by clicking the **Freeze** button.

The Icon Output Window

Since every task in Metview is a sequence of operations on icons, system and error messages are usually generated following an icon operation. Another destination for messages is the *icon output window*.

The icon output window records the system feedback that refers to this icon alone as well as any console output from the icon (for Macro-class icons). To open this output window :

- Click-right the icon and select option **Output**.

This opens a text window where you will find the relevant message or output (in the case of macro icons).

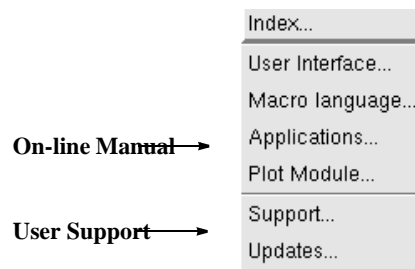
The output window is only available after the first message is produced (does not open as blank).

Also if you edit and save the icon, any existing text is wiped out and the **Output** option becomes unavailable again. The same happens if you carry out a new operation on the icon that generates new messages - the resulting new messages will overwrite the old ones.

If you need to look at a history of the messages generated by a particular icon you need to check the Message window, where all messages remain through the Metview session except if erased by the user.

The Metview Help

The Metview Help is available from the Help menu of the desktop bar. The menu is shown below :



The **Metview Help** system consists of two main components :

- On-line Metview Manual
- User Support system

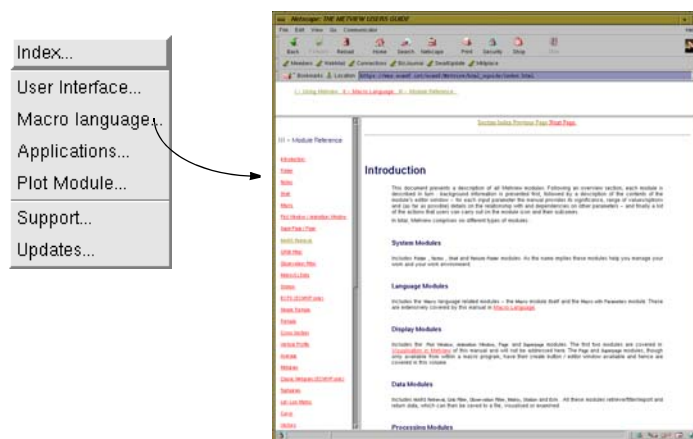


Figure I-71 : Metview On-line Manual. Netscape is launched automatically when you click one of the options in the Help menu (Macro Language in this example).

On-line Metview Manual

This is essentially this manual in HTML viewed in a browser (Netscape or other UNIX compatible). Access to this manual is arranged in five entries of the Help menu. Each entry when selected launches Netscape with the URL for the corresponding on-line HTML manual :

Index - Access to the on-line global table of contents

Using Metview, Macro Language, Applications - Access to the on-line manual sections corresponding to the identically named volumes making up the printed manual

Visualisation - Access to the on-line visualisation description contained in this volume

User Support

The built-in system for user support launches the Metview Mail tool with a subject and a destination address already provided. The address reaches the Metview support team. The subject identifies the message as a Metview Bugs Report.

Users should mail a description of the problem and the icons involved, using the icon mailing feature of the Metview Mail tool described in "Mail" in "The Metview Tools" on page I- 129. The user support team will be able to extract the icon(s) and replicate the actions that lead to the problem you are experiencing.

Note that the advice or solution to the problem may be provided to you exclusively via your standard e-mail tool. However, if the reply contains icon(s) (maybe your own icons after some amendment), then you will be able to read the reply and install associated icons using the Metview News tool (see "News" in "The Metview Tools" on page I- 130)

The Metview Tools

Metview includes a set of tools that aim to ease the user work; they comprise a **Mail** tool to allow communication and icon sharing between users, a **News** tool for users to install worked out examples and icons sent by colleagues, a **Monitor** to provide information on the current active processes, a Stations database search tool and an external application (Vis5D) which has been integrated with Metview.

The set of Metview tools is available from the **Tools** menu of the desktop menu bar :



Mail

Metview Mail is a built-in mail tool, developed to provide seamless communication between all Metview users and between these and user support and development teams, irrespective of the location of their work place.

Metview Mail can be used to send simple messages to whoever you wish (not just other Metview users). However, the distinguishing feature of Metview Mail is the ability to send icons as attachments (icons are automatically uuencoded), which can then be extracted to the Metview environment of the destination address. This enables one user to demonstrate work results to users in other offices or countries. It may be used to build cooperative efforts, sending bits of work back and forth in order to build a complex task.

Sending mail

Start the Mail tool by selecting Mail from the Tools menu. The Mail window has a text pane and an icon drop pane. Type your message in the text pane and drop the relevant icons in the icon drop pane - the icon names will appear here. If you need to remove an icon from the drop pane, click its name and then the Remove button - see Figure I-72 for details. Once everything is done, click the Send button.

Any number of icons can be mailed, and icons embedded in a mailed icon are also mailed. Mailing a folder icon will also mail all icons contained in the folder (which is a convenient and effective way to send groups of icons). Large size GRIB file icons may be sent more effectively by other means.

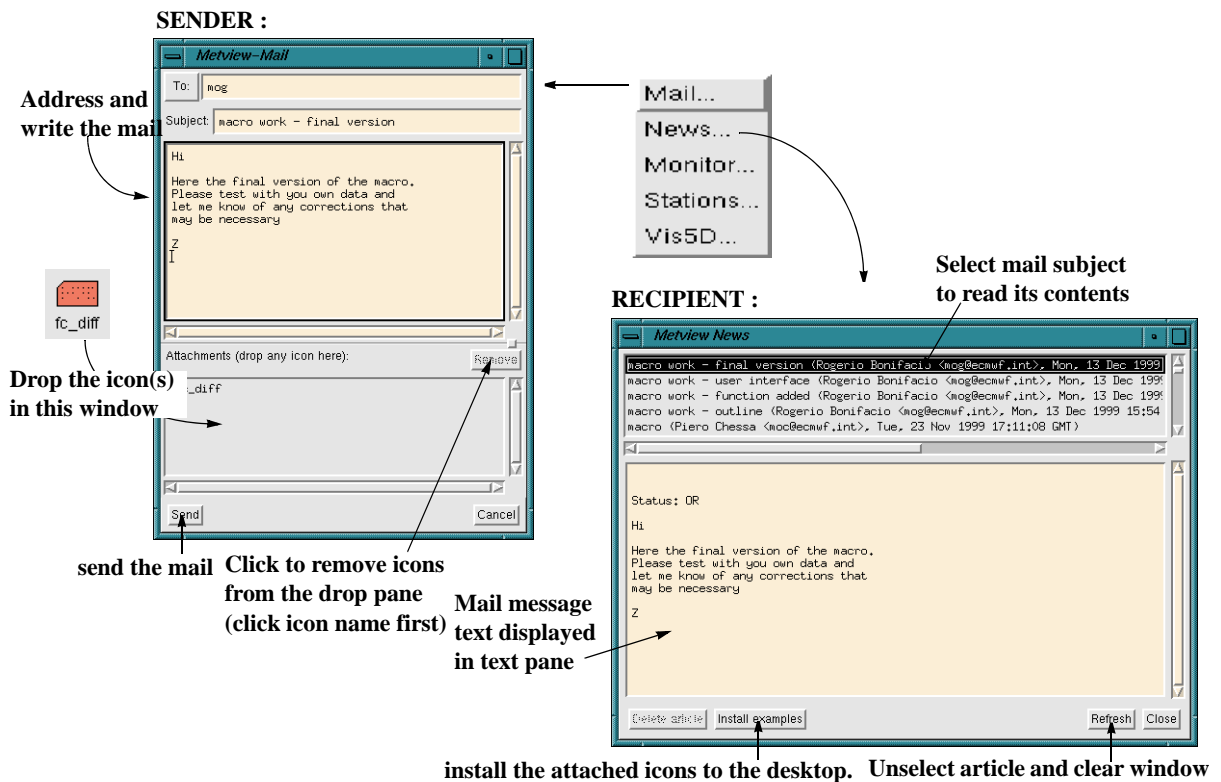


Figure I-72 : Sending and receiving messages and icons in Metview. Use Metview Mail to send and Metview News to receive/install.

Receiving mail

The Metview Mail tool can only be used to send items. Reading incoming mail is done either with your usual UNIX mail application, or in case the incoming mail message has attached icons you need to use the **Metview News** tool (see "News" on page I- 130) since it allows you to unpack and install automatically attached icons sent by means of Metview Mail : if you examine the contents of a message containing an attached icon on your UNIX mail tool, you will see the string (tag line) :

```
--METVIEW-ICONS
```

below the message text. It is this string that allows Metview News to identify the message as a Metview article within the mail spool. It is up to you to check the mail box or the Metview News tool for incoming mail, Metview does not provide an automatic "new mail" warning.

Some technical details

Metview looks for an incoming mail folder using environment variable \$MAIL. If \$MAIL has not been set (normally set during Unix login sequence), then /var/spool/mail/\$LOGNAME is used.

Although you can send icons as if they were attachments, there are no explicit attachments in Metview mail : Metview icons are written into a tar file which is first compressed, and the compressed tar file is then written as part of the mail message, using uuencode. This uuencoded compressed tar file is indicated by a tag line (--METVIEW-ICONS).

Thus the whole mail is a single text, composed of your comments (typed in the upper window), a tag line, and the text representing the uuencoded compressed tar file (icons dropped in the lower window). You can test how this works by sending to yourself some icons, and checking how the message looks like with your normal mail reader.

News

Metview News is a tool that stores and lists articles and the mail messages which have icons attached - *only mail messages with icons attached are detected by the News tool.*

By "articles" we mean predefined examples, new features and information, technical tips, frequently with icons attached. The tool allows you to read the contents of the articles and mails and to install the associated/attached icons. To launch the **News** tool either :

- choose News from the **Tools** menu
- or
- *Execute* the What's New icon on the main desktop

The News tool interface lists article titles and mail message subjects in a top window and displays the text of the selected article or mail message in a lower read-only text field.

To read an article or mail, click the article title or mail subject on the top field to display the text on the lower field. If there are attached icons, the **Install Examples** button becomes available - click it to install the icons - see Figure I-72 and below for details.

To unselect the article, clear the text window and retrieve new messages, click the **Refresh** button.

The pre-installed News article 'Reading Metview Mail' provides detailed information about configuring your mail folders if icons that have been mailed to you are not visible from within the News tool.

Installing examples :

- Click the **Install examples** button and Metview creates a folder in the main desktop whose name follows the syntax Examples from <article title>, e.g. installing the article Image manipulation examples will create a folder named Examples from Image manipulation examples
- After installing the example, open the installed folder to reveal a text file icon and other icon(s), probably a single sub-folder icon. The text file icon contains the text displayed in the News window. The sub-folder icon contains the icons necessary to follow the example through

Installing mailed icons :

- Click the **Install Examples** button and Metview creates a folder whose name follows the syntax : Mail from <mail subject> (<sender address><date>), e.g. Mail from macro work - final version (xyz@ecmwf.int, Fri, 21 Jun 2000, 12:30:15, +0100) . What is happening is that Metview looks for the tag line --METVIEW-ICONS, and uuencodes the message, uncompresses it, and untars it into your Metview folder.
- The contents are a simple text file icon with the text of the message you have viewed in the News window and the mailed icons. When you mail an icon its original path is preserved so you may find the icons several folders deep, each folder having the same name as its original location

Monitor

Monitor provides you with the means to assess the progress of your tasks and with a degree of intervention upon running tasks. It can be launched and its window iconified on the workstation screen and brought up whenever need or curiosity arises.

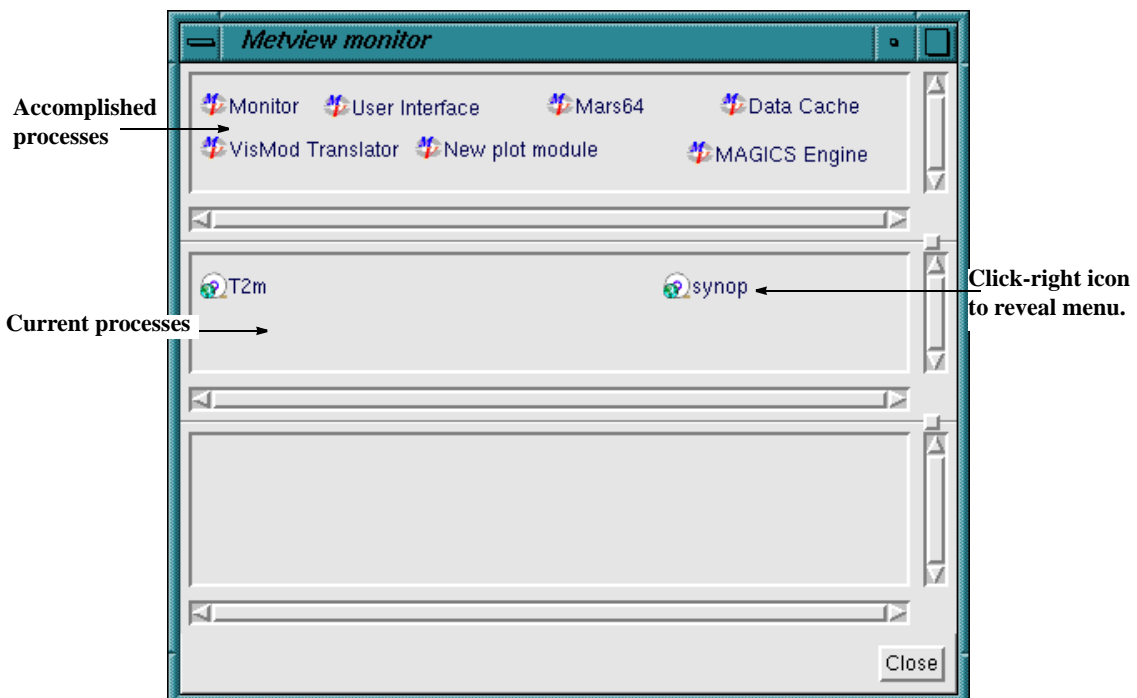


Figure I-73 : Example of the Monitor window mid-way through a simple visualisation. The current processes are retrievals from the database. You can intervene on them (aborting and information) by means of a click-right icon menu.

The **Monitor** window is divided in three panes (see Figure I-73) :

- The bottom pane will contain icons for processes which are waiting to start.
- The middle pane contains icons for current processes.
- The top pane contains icons for processes which have been accomplished (these are UNIX processes which are kept running).

For an example, run a visualisation of your choice with the monitor window open - in this way the flow of requests and process executions behind a visualisation can be appreciated. The content of the panes changes according to the progress of the task, thus allowing you to assess it by noting which icons are displayed in which pane.

Figure I-73 shows an example of a monitor window mid-way through a couple of retrievals. Data retrieval is the usual process that resides long enough in the middle pane to allow or require user intervention - intervening on a process depends on how fast it is dealt with by Metview. A simple process on a fast workstation may be over before you can intervene. User intervention is enabled by means of a click-right icon menu with the following choices :

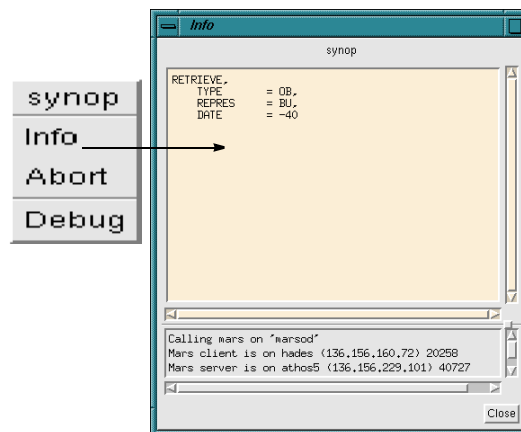


Figure I-74 : Outcome of Info option in a running MARS request process. It shows the MARS directives corresponding to the process and information on its status.

Info - This option provides information on the process. For a process in the top pane, the information provided is just the machine hostname, the user's workstation user name and the PID (UNIX process ID). For a process in the middle pane, the information provided is the description of the process in terms of MAGICS or MARS directives (see Figure I-74)

Abort - This option kills the process to which it applies. Abort allows you to kill a process which may be hung, or which you do not want to run after all - note however that a visualisation involves several separate processes and you may have to abort several.

If you abort the Data Cache process you flush all the data stored in memory (data requests are cached to minimise retrievals).

Debug - This option is only active to the developers team.

Stations

Stations is a tool to search a database of meteorological stations. Selecting **Stations** from the **Tools** menu brings up the Metview Stations Database (MSD) tool, whose interface is shown in Figure I-75. The main features are :

- a top menu bar with the items File, Plot and Help. The only item in File is Close. Plot allows you to plot the location of retrieved stations.
- a top icon pane and a lower (read-only) text pane
- an area where you specify the type and values of the search parameters
- a bottom area where messages are printed and where the **Close** button is placed

Typically a session with the **Stations** tool involves a search according to user defined criteria, followed by some action on the search results (plotting locations and/or save the search results to a desktop as Station icons).

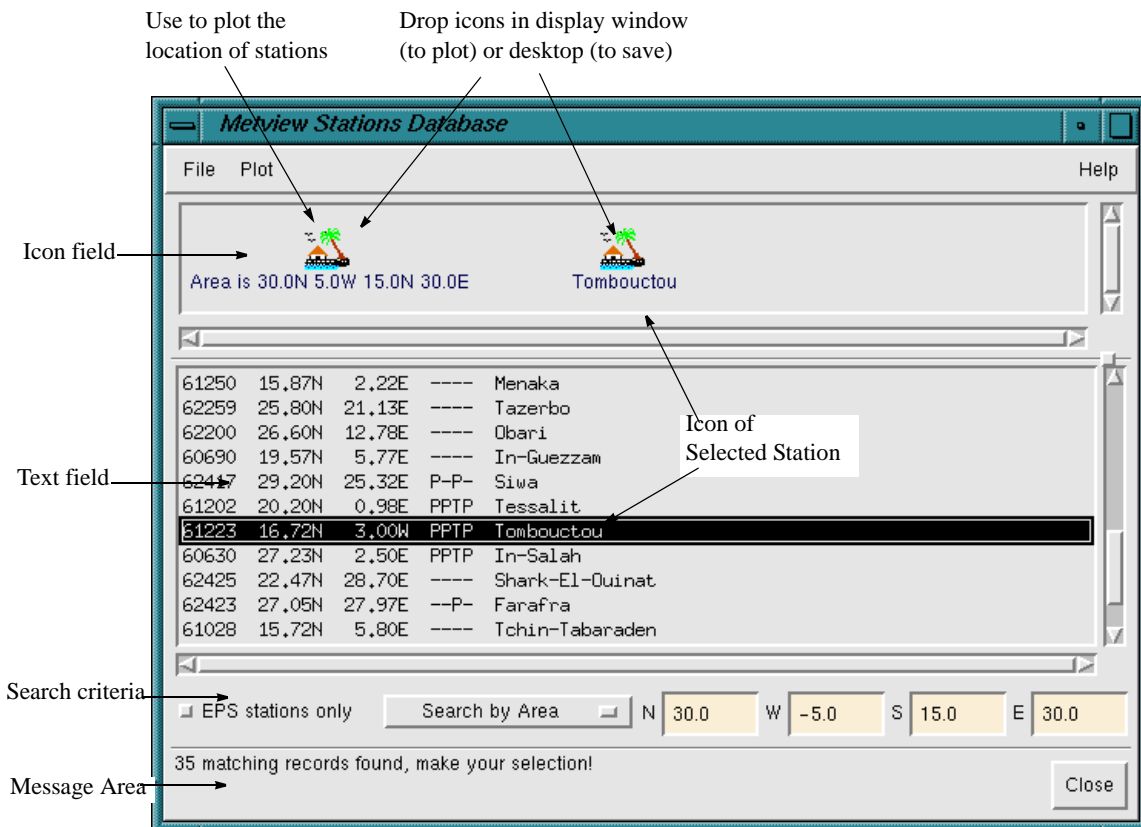


Figure I-75 Example of the Metview Stations Database window after a successful search. In this example, the user has searched By Area, obtaining the meteorological stations within the Sahara Desert. The result of a search looks the same irrespective of the type of criteria employed.

Database search

To search the Stations Database, first select a search criteria from the option menu. The search criteria include **By Name**, **By Identifier**, **By WMO Block**, **By Position** and **By Area** :

By Name - this search uses a single field where you specify a text string of n characters. The search is activated when you hit return with the mouse pointer inside the field. The specified string of n characters is compared to the first n characters of the station names in the database and the matches are printed in the text field. The elements printed out are the WMO identifier, the latitude, the longitude and the station's name. Note that the spelling of the station's name is in accordance to WMO (which tries to match the original language spelling). Hence, an english speaking user should look for Kobenhavn or Porto rather than Copenhagen or Oporto. An icon is returned to the top icon pane with the name Name starts with 'xxxx'.

By Identifier - this search uses a single field where you specify a number of either 2 or 5 digits. A 5 digit number is taken as a station identifier, a 2 digit number is taken as a WMO block identifier. The search returns the station which matches the station identifier or those that reside within the specified WMO block (see below). The output icon in the top pane is named Identifier is 'nnnnn'.

By WMO block - this search uses a single field where you specify a number of up to 2 digits (if more than two, it uses the last two). The search returns the stations residing in the specified WMO block. The output icon in the top pane is named WMO block is 'nn'.

By Position - this search uses three fields. The first two are the (approximate) latitude and longitude of the station and the third is a tolerance threshold in degrees. South latitudes and West longitudes should be given as negative numbers. The search returns all stations within +/- tolerance around the specified coordinate. The output icon in the top pane is named Position is xx.x yy.y (+-number).

By Area - this search uses four fields, where you should input the geographical limits (lat/long) of a rectangular area. South latitudes and West longitudes should be given as negative numbers. The search will return all stations within this area. The output icon in the top pane is named Area is top_lat left_lon bott_lat right_lon.

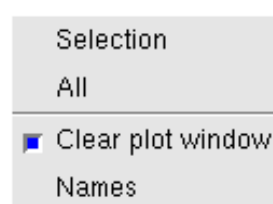
EPS Stations Only - An additional search criteria is to retrieve only EPS Stations - click the check button to the left of the criteria type menu. These stations form a group for which EPS data (cloud cover, precipitation, 10m wind speed and 2m temperature) has been interpolated at the station location and stored as BUFR files, for quick plotting of station EPS Metgrams.

Once your search is finished you should get a display like that shown in Figure I-75. From here, you may choose to plot the stations in a display window and / or transfer the retrieved stations to a desktop as Station icons.

If you want to act on a single station out of those you retrieved, you must *select* it by clicking its register in the text field. The register becomes highlighted and a station icon for the selected station appears in the icon field (see Figure I-75). You can only select one station at a time. Selecting another station will replace the one previously selected.

Plotting stations

You can plot the geographical location of the stations obtained from the search using the options in the **Plot** menu or by dropping an icon from the icon pane inside a display window frame. The **Plot** menu includes the following options :



To plot, choose one of :

Selection - plots only the selected station; if none is selected, the **Selection** option is not available

All - plots all found stations

A display window shows up with markers at the geographical locations of the stations. Once the display window is up, you may instead drop the station icons from the icon pane into the display window frame.

The last two other options specify plotting modes :

Names - plots the station names in the plot window. By default, names will not be plotted (the option is unchecked).

Clear plot window - if you make another retrieval of stations and plot the new result, the previously plotted stations are erased by the new ones if this option is toggled on; otherwise the new stations are added to the previous plot. This does not apply if you plot the new station locations by icon drop - previous plotting results will remain displayed in the display window frame. This option is active by default.

After the first plot, you may use the display window facilities to zoom in/out and change projection and/or area. Alternatively, you may execute the display window of your choice and *drop* the icons in there. If you change the display window layout to several independent frames you will be able to plot different station retrievals in each frame.

Exporting station icons

You may *drop* any of the icons present in the icon field into one of your work spaces. The icons can then be used as input to other applications.

II- MACRO LANGUAGE

MACRO LANGUAGE FUNDAMENTALS

This volume describes the Metview Macro language. A macro language was part of the first design specification of Metview. It is designed to perform data manipulation and plotting from within the Metview system environment. A language is the best "user interface" to describe very complex sequences of actions, particularly if the flow of action is conditional. It also provides a common means to express the mathematical formulae used when performing data manipulations.

The Metview Macro language was designed to be as easy to get started with as a script language (e.g. UNIX shell) and as powerful as a modern computer language. To be as simple as a shell language implies that no variable declarations or program units should be required. This feature is achieved through the implementation of typeless variables, a benefit of object-oriented languages. To be as complex as a computer language implies support for variables, flow control, functions and procedures, I/O and error control.

The Metview Macro language provides an easy, powerful and comprehensive way for a researcher to manipulate and display meteorological data. It extends the use of Metview into an operational environment as it enables a user to write complex scripts that may be run with any desired periodicity.

This manual assumes the reader is familiar with the basic concepts of a programming language (e.g. concepts of "variables" or "functions").

Basic Syntax

Identifiers

Identifiers are words used to name variables and functions. A valid identifier is a word that starts with a letter or an underscore, followed by any number of letters, digits or underscores.

Note that identifiers are case sensitive.

Reserved keywords

The following words are reserved and cannot be used as identifiers:

and by case do else end export extern for function global if import in
include inline loop nil not object of on or otherwise repeat return task
tell then to until when while

Comments

To write comments, use the character #. Any text from this character to the end of the line is ignored. Each line to be commented must be preceded by the # character, i.e. C-like multi-line comments are not allowed.

```
# Now plot the field
plot (Z500) # using default contours
```

Operators

The operators are, in order of decreasing precedence :

-	Negative		
+	Addition	-	Subtraction
*	Multiplication	/	Division
^	Power		
&	Concatenation		
>	Larger Than	<	Smaller Than
>=	Larger or Equal	<=	Smaller or Equal
=	Equal	<>	Not Equal
and	Conjunction	or	Disjunction
in	List membership	notin	List membership

All operators except power (^) are left-associative which means that

$x + y + z$ is evaluated as $(x + y) + z$

and that

$x ^ y ^ z$ is evaluated as $x ^ (y ^ z)$

You can use brackets to modify the order of computation.

Variables

Creating Variables

Variables are created when assigned for the first time. They take the type of whatever is assigned to them.

```
# Variable Z500 is a string
Z500 = "500 hPa geopotential"

# Variable Z500 is now a fieldset
Z500 = retrieve(...)
```

Local variables

Variables are local by default, i.e. only "visible" within the function where they are defined, *not* in any of its embedded functions. Variables are passed "by value" to functions - i.e. their value outside the function they are passed to is left unaffected. You can modify this - see "Modifying the scope of variables" on page II- 5.

Global variables

Global variables are "visible" both within the function where they are defined *and* in all its embedded functions. This allows programmers to write stand-alone macros using their own global variables, and to give them to a user as a library function without causing any clash with this user's own global variables.

Global variables keep their values between function calls. They are initialised with the value 0 (zero), or with an expression :

```
global ident
global ident = expression
```

The expression is evaluated before the execution of the first instruction in the block where the global variable is declared. If a global variable is declared in a function, its initialising expression is evaluated only once, the first time the function is called. Some global variables are predefined (but you can change their value) :

Name	Type	Value
e	number	2.71828
pi	number	3.14159
tab	string	ascii 9 (tabulation)
newline	string	ascii 10 (carriage return)
geo_missing_value (this should not be changed)	number	3e+38

Lifetime of variables

A variable is created simply by assigning something to it. While the variable is "in existence" it will use up some memory and possibly disk space. This memory is freed and disk space released when the variable is destroyed. A variable is destroyed either :

- explicitly by the programmer - to destroy a variable simply set it to zero
- automatically by the system - this happens when the variable becomes out of scope, e.g. when the end of the function where it was defined is reached

In the following example, the disk space used by the variable a is kept until the end of the macro, because there is no way to know that a is not going to be reused :

```
a = retrieve(...)
plot(a)

# Lots of lines not using the variable a
(...)
```

However, you can explicitly destroy the variable a by writing :

```
a = retrieve(...)
plot(a)
a = 0 # deletes any retrieved data

# Lots of lines not using the variable a
(...)
```

Another way to do this would be to use a function - because the variable is local to the function, the variable is destroyed at the end of the function, freeing memory and disk space :

```
function plot_a
  a = retrieve(...)
  plot(a)
end plot_a

# Main routine
plot_a()

# Lots of lines not using the variable a
(...)
```

Scope of variables

The scope of a variable is the length of code during which the variable is in existence. If the code contains functions the variable may be out of scope within the function but this depends on whether it was declared as a global variable.

The best way to understand scope of variables is to work through a schematic example using a macro containing a variety of functions as shown below - the macro text is shown on the left and the list of variables in scope on the right :

Macro text	In-scope variables
global g1 x1 = 9	g1 g1,x1
function f1 global g2 x2 = 8	g1,g2 g1,g2,x2
function f2(p2,p1) global g3 x3 = 7 end f2	g1,g2,p2,p1,g3 g1,g2,p2,p1,g3,x3
... x4 = 8 ...	g1,g2,x2 g1,g2,x2,x4 g1,g2,x2,x4
end f1 ... x5 = 8	g1,x1 g1,x1,x5

Users can modify the scope of local variables relative to functions by importing and exporting them (next).

Modifying the scope of variables

Users can modify the scope of local variables by importing and exporting them - this makes them be in scope inside the function. When you import a variable from an outer block, a local copy is made and any modifications made to the variable are not made to the original variable. If you export the variable first, it will be actually shared, and all modifications made to it in the inner block will also be done in the outer block.

```

a = 9
b = 10
export a

function foo
  import a
  import b
  print("Inside foo 1, a=",a," b=",b)
  a = 1
  b = 1
  print("Inside foo 2, a=",a," b=",b)
end foo

print("Before foo, a=",a," b=",b)
foo()
print("After  foo, a=",a," b=",b)

```

this code will print:

```

Before foo, a=9 b=10
Inside foo 1, a=9 b=10
Inside foo 2, a=1 b=1
After  foo, a=1 b=10

```

Null variables and the nil keyword

The `nil` keyword represents a variable with no value *and no type*. It can be useful for building lists and for checking the success of a data retrieval function. A `null` variable is one which has either the value of `nil` or the value of 0 (zero). Note that 0 (zero) is not the same as `nil`, since zero has a type (number).

Program Control

In this section we cover the macro language instructions that enable user control over the program flow. You are most likely familiar with most or all of these from other programming languages - however, FORTRAN programmers should note the absence of the `goto` instruction.

Tests - the if (and if/else) statement

Used for decisions based on testing the result of an expression. A value of 0 means false, any other value means true. The macro language supports three kinds of `if` statements:

```

if a = b then
  print('a and b are equal')
end if

if a = b then
  print('a and b are equal')
else
  print('a and b are different')
end if

if a > 0 then
  print('a is positive')
else if a < 0 then
  print('a is negative')
else
  print('a is null')
end if

```

Note that the macro language ignores any newlines between an 'else' and a subsequent 'if', always interpreting this as 'else if'. The following piece of code:

```

if ... then
  ...
else
  if ... then
    ...
  end if
end if # wrong - syntax error

```

should be written more clearly as:

```

if ... then

```

```
...
else if ... then
...
end if
```

Tests - the when statement

The `when` statement is equivalent to a list of `if` and `else if` statements. The code following the first true expression is executed and the `when` statement exited.

```
when
  a > 0 :
    print('a is positive')
  end
  a < 0 :
    print('a is negative')
  end
  a = 0 :
    print('a is null')
  end
end when
```

Tests - the case statement

The `case` statement compares an expression to a number of other expressions. The code following the first match is executed and then the `case` statement is exited. If no match is found, an optional `otherwise` clause is executed.

```
case type(x) of
  'number' :
    print('x is a number')
  end
  'date' :
    print('x is a date')
  end
  otherwise :
    stop('Unsupported type')
  end
end case
```

Loops - the for statement

In a `for` statement the variable takes all values between two expressions in steps of a specified increment. A `for` loop can be performed on any type `T` where `T+number` is a `T` and `T-T` is a number (e.g. dates or numbers)

```
# Prints d in steps of two days
for d = 1997-09-07 to 1997-09-18 by 2 do
  print(d)
end for
```

Note that the number of iterations in a `for` loop is determined at the start of the loop, and not recalculated during its execution.

Loops - the while statement

The `while` statement is executed only if the test is true.

```
n = 1
while n <= 10 do
  print(n)
  n = n + 1
end while
```

Loops - the repeat statement

The `repeat` statement is always executed once. It stops when the test results in false.

```
n = 1
repeat
  print(n)
  n = n + 1
until n > 10
```

Loops - the loop statement

The `loop` statement will make a variable take (one by one) all the values contained in a list. Loops can be done on fieldsets, geopoints, or more generally on any type that supports the `count()` function and indexing.

```
loop n in x
  print(n)
end loop
```

is equivalent to:

```
for i = 1 to count(x) do
  n = x[i]
  print(n)
end for
```

MACRO DATA TYPES

In the following sections the Metview Macro types are presented and defined. Currently, the Metview Macro types are :

- Numbers
- Dates
- Lists - arrays of variables of single or mixed type
- Definitions - lists of named variables
- Fieldsets - variables to hold model fields (GRIB format)
- Observations - variable to hold BUFR data (meteorological observations)
- Geopoints - variables to hold (irregular) point data
- NetCDF - variable to hold general numerical data
- Images - variables to hold satellite images (modified GRIB format)
- Arrays (Vectors and Matrices) - for numbers only

Finding the Type of an Expression

To find the type of a variable or expression use the function `type()`, which accepts any variable or expression as input. The type is returned as a character string which can be printed or used in comparisons between variables :

```
# Returns the type of the following expressions
x = 9
print(x, ' is a ' , type(x))
x = 2000-01-01
print(x, ' is a ' , type(x))
x = "abc"
print(x, ' is a ' , type(x))
x = [1,"abc",2]
print(x, ' is a ' , type(x))
```

this will print:

```
9 is a number
2000-01-01 00:00:00 is a date
abc is a string
[1,"abc",2] is a list
```

Strings

For a list and details of functions and operators on strings, see "Functions and Operators on Strings" on page II- 129

Strings are sets of characters. They are unlimited in length. String literals are defined using single or double quotes :

```
s = "foo"  
s = 'bar'
```

You can concatenate strings with numbers, dates and other strings. The result is always a string :

```
parameter = "z"  
level = 500  
file_name = parameter & level & ".grib"  
print (file_name)
```

will print the following string :

```
z500.grib
```

You can retrieve substrings using the `substring()` command :

```
print (substring ("Metview", 4, 7))
```

will print the following string :

```
view
```

You can split strings into substrings, using the function `parse()`. This takes two strings as input : the first is the string to split, the second is a string consisting of all the separators; it returns a list whose elements are the individual strings resulting from the split (or tokens):

```
n = parse("z500.grib", ".")  
print ("name = ", n[1], " extension = ", n[2])
```

will print the following string :

```
name = z500 extension = grib
```

Note that the separator character(s) are not part of the output token list.

You can create multi-line strings, using the `inline` keyword :

```
s = string inline  
This is a multiline string, with all sorts of characters such as double  
quotes (") or single quotes (') which are ignored in the inline context.  
end inline
```

Numbers

For a list and details of functions and operators on numbers, see "Functions and Operators on Numbers" on page II- 125. No distinction is made between integer or real numbers. All numbers are internally coded as double precision floating point real.

Dates

For a list and details of functions and operators on dates, see "Functions and Operators on Dates" on page II- 133.

The macro language defines dates as a type. A date contains information about the year, month, day, hour, minute and second.

Creating

Dates can be created as literals using the following syntax:

```
YYYY-mm-dd  
YYYY-DDD
```

Where `YYYY` is a four digit year, `mm` is a two digit month, `dd` is a two digit day, and `DDD` is a three digit julian day (day ordinal in the year, the 1st of January being day 1). Two digit years (`YY`) were valid but are now deprecated and illegal in some requests (e.g. MARS retrievals) and functions.

To introduce time of the day information, use the following syntax :

```
HH:MM  
HH:MM:SS
```

where `HH`, `MM` and `SS` are respectively the hour, minute, and second, using two digits. This information is simply appended to the date specification, e.g.

```
my_date = 2000-01-04 09:50:24
```

Alternatively, you can create dates from numbers, using the function `date()`. This function works with negative (and zero) numbers or with 8 digit positive numbers.

The function `date()` interprets negative numbers as days before today and a zero value as today, e.g.:

```
today      = date(0)  
yesterday = date(-1)
```

and it interprets an 8 digit number as `yyyymmdd` :

```
d1          = date(20000104)
print("date is : ", d1)
```

This prints :

```
date is : 2000-01-04 00:00:00
```

This clearly shows that the `date()` function creates dates where the hour, minute and second components are zero. To create a full date, use decimal dates or the functions `hour()`, `minute()` and `second()`. All following examples :

```
d = date(20000104.41)
d = 2000-01-04 + 0.41
d = 2000-01-04 + hour(9) + minute(50) + second(24)
```

assign the same date to variable `d`. When printed (`print(d)`) it gives :

```
2000-01-04 09:50:24
```

To create a date variable holding both the current date and time, use the `now()` function :

```
d = now()
print ('Now the date and time are ', d)
```

This gives :

```
Now the data and time are 2005-06-20 16:40:31
```

Dates in Metview MARS requests

The MARS language (for retrieval of data from ECMWF archives) automatically converts numbers to dates. This applies also to the Metview icons. For consistency, the macro language also accepts dates specified as numbers without the need to use the `date()` function :

```
r = retrieve(date: -1, ...)
r = retrieve(date: 20000104, ...)
```

Users should bear in mind that when passing a date to Metview requests that interface with MARS, such as `retrieve()`, `obsfilter()` or `read()`, the hour, minute and second information are lost, as MARS can only handle integral dates. Thus the time has to be passed as an extra parameter :

```
d = 2000-09-07 12:00:00
x = retrieve(
    date : d,
    time : hhmm(d),
    ...)
```


Converting dates to strings and numbers

General conversion

Dates can easily be converted to strings or numbers in Metview Macro. This conversion is handled by the functions :

```
string(date, format)
number(date, format)
```

They both take a date as the first argument and a format specifier as a (facultative) second argument. The simplest conversion does not use the format specifier explicitly :

```
dd = date(20000104.41)
ds = string(dd)
dn = number(dd)

print(type(dd), " : ", dd)
print(type(ds), " : ", ds)
print(type(dn), " : ", dn)
```

The print comands of this short piece of code yield :

```
date   : 2000-01-04 09:50:24
string: 2000-01-04 09:50:24
number : 20000104
```

Note that although the first two variables print identically their type is different - you cannot use the string variable `ds` in functions requiring a date variable. Note as well that the function `number()` returns an integer, discarding the time stamp. When you do not use the format specifier string as a second argument, a default one is implicitly used - this default is customisable, see "Configuring date formats" below.

Converting date components

A date is a multidimensional variable in the sense of being composed of year, month, day, hour, minute, second. You may need to extract one (or more) of these components from a given date and to express these components in a variety of ways, e.g. you may need day of the month or day of the year, number of the month, month as a string, etc.,

Both the extraction of a date component and its expression in a variety of formats are handled by the `string()` and `number()` functions as well - their second argument (format specifier) which determines which component is extracted and in which format. E.g. if you need the year of a date as a four digit number :

```
dd = date(20000104.41)
yrn = number(dd, "YYYY")
yrs = string(dd, "YYYY")
print(type(yrn), " : ", yrn)
print(type(yrs), " : ", yrs)
```

the format specification ("YYYY" in this case) is always a string given as the second argument to the `number()` or `string()` functions. The output of the above is :

```

number : 2000
string: 2000

```

A full list of the format specification strings is given below using the date of the above examples - 09h50m24s of the 04th of January 2000. The available format specification strings used in the `string()` and `number()` functions when applied to this date yield :

- `YY` gives 00 (string) or 0 (number)
- `YYYY` gives 2000
- `m` gives 1
- `mm` gives 01 (string) or 1 (number)
- `mmm` gives Jan (string only)
- `mmmm` gives January (string only)
- `d` gives 1
- `dd` gives 01 (string) or 1 (number)
- `ddd` gives Tue (string only)
- `dddd` gives Tuesday (string only)
- `D` gives 4 (4th of January = julian day 4; leap years accounted for)
- `DDD` gives 004 (string) or 4 (number)
- `H` gives 9
- `HH` gives 09 (string) or 9 (number)
- `M` gives 50
- `MM` gives 50
- `S` gives 24
- `SS` gives 24
- Any other character is copied as such

All of the above are applicable in a conversion to string. Only those which produce a numerical format are valid for a conversion to number as indicated (e.g. `m` is applicable, `mmm` is not).

You can mix your own bits of text with the above string formats in order to print full dates in a reader friendly way. e.g.

```

dd    = date(20000104.41)
sdate = string(dd, "dddd, ddth mmmm yyyy")
print (sdate)

```

will output :

```

Tuesday, 04th January 2000

```

Format specifiers can also be used to perform date calculations in a very efficient way :

```

today      = date(0)
last_day   = date(string(today, "yyyy") & "1231")
n          = number(last_day, "D") - number(today, "D")

```

```
print("number of days to the new year : ", n)
```

This outputs the number of days from today to the end of the current year - you set up the date of today and of the last day of the year as date variables, express them as Julian day numbers (using `number(date, "D")`) and subtract them to obtain the required output. Note also that subtracting one date from another gives the number of days between the dates. The above example could be rewritten:

```
today      = date(0)
last_day  = date(string(today,"yyyy") & "1231")
print("number of days to the new year : ", last_day - today)
```

Configuring date formats

You have a degree of control over the date formats used by Metview Macro. You can:

- modify the default format specification string for conversion to string and number - the default format specification string for conversion to string is "yyyy-mm-dd HH-MM-SS" and for conversion to number is "yyymmdd" (yielding a string "2000-01-04 09:50:24" and number 20000104, respectively, for the example we have been using).
- replace the default english string date components (names of the week days and of months) - the default month names are January, February, ..., while the default week days' names are Monday, Tuesday, You can replace these by those in any language of your choice, e.g. Janvier, Janeiro, Enero (no accents though!).

Both the default format specifications and default string date components are specified in the Preferences option of the File menu in the menubar of any Metview desktop. When you select this option the Preferences editor is launched (see Figure II-1) - simply type in the required default format specification and/or the month/weekday names in the language of your choice. When you save the result they will come into use immediately.

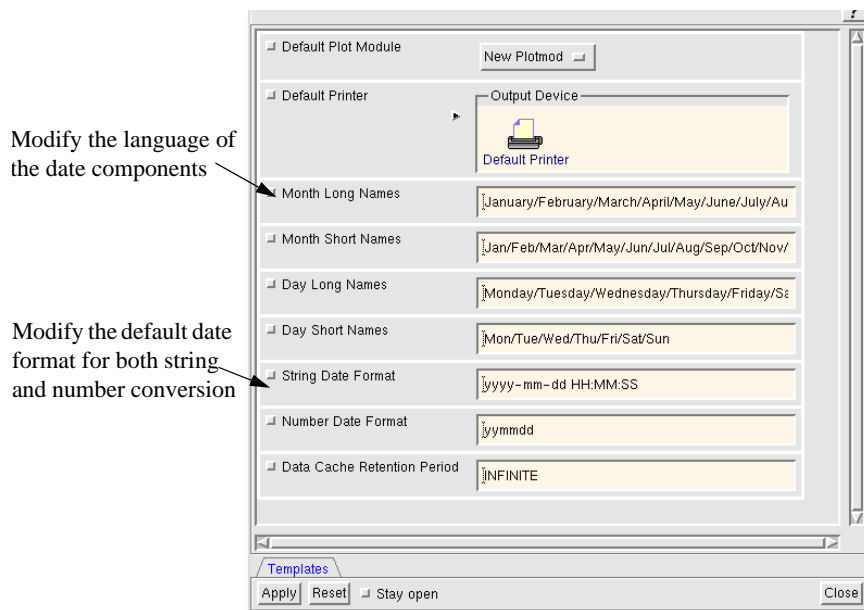


Figure II-1 : The Configuration editor window. To modify the default string or number date representations, type in the required formats in the text fields. If you need dates in a language other than english, enter here the names of the months and days as printed by Macro.

Loops with dates

It is possible to do loops with dates using a for loop, with increments of any number of days or of fractions of day :

```
# using default increment (1 day)
for d = 1997-09-01 to 1997-09-10 do
  (...)
end for

# using a non default increment of 2 days
for d = 1997-09-01 to 1997-09-10 by 2 do
  (...)
end for

# using a non default increment of 6 hours
for d = 1997-09-01 to 1997-09-10 by hour(6) do
  x = retrieve(
    date : yymmdd(d),
    time : hhmm(d),
    ...)
  (...)
end for
```

Lists

A list is an ordered, heterogeneous collection of values. Lists are not limited in length. A list element can be of any type, including another list. Lists are built using square brackets or the `list()` function. The first element of a list has index 1.

```
l = [3,4,"foo","bar"]
l = list(3,4,"foo","bar")

or

l = [2,3,[3,4]]
l = list(2,3,[3,4])
```

Lists are a convenient way to build and handle non-numeric arrays. You may use them for numeric arrays, though the macro language provides for both vectors and matrices of numbers. The following piece of code will print each element of the list `list_x` in turn:

```
for i = 1 to count(list_x) do
  print (i, " : ", list_x[i])
end for
```

If building a list from scratch in a loop, you should initialise the list to the `nil` value and add new elements using the concatenate operator (`&`); the new elements to be added to the list must be expressed as single element lists since the concatenation operation requires operands of the same type :

```
# Creates a list of dates
date_list = nil
for d = -10 to -1 do
    date_list = date_list & [date(d)]
end for
```

A number of functions are available for handling lists - accessing individual elements, extracting sub-lists, sorting lists, testing for presence of an element in a list - see "Functions and Operators on Lists" on page II- 137.

Areas

Areas are defined using a list of four numbers for the north, west, south and east boundaries. Northern latitudes and Eastern longitudes are positive, while Southern latitudes and Western longitudes are negative. Note that data retrieval modules (and their macro requests) require the latitude and longitude of the top left corner followed by those of the bottom right corner, while plot and animation window modules require latitude and longitude of the bottom left corner followed by those of the top right corner. So to represent Europe in a `retrieve()`, you should specify :

```
x = retrieve(
    ...
    area : [75,-12,35,42],
    ...)
```

while for a `map_view()` you should specify :

```
x = map_view(
    ...
    area : [35,-12,75,42],
    ...)
```

Definitions

A definition is a collection of named items, each item being a *member* of the definition. Each member must be assigned a value which can be a number, a string or a variable of any type, including other definitions. You create a definition by assigning it to a variable using the following syntax :

```
d = (name1:value1, name2:value2, ...)
```

e.g.:

```
a = (x : 1, y : 2)
b = (a, p : "T2", f : data)
```

In the example above, `a` contains members `x` and `y` which are assigned the numerical values 1 and 2; `b` contains `x`, `y`, `p` and `z` which are respectively assigned the numerical values 1 and 2, the string "T2" and the variable `data` (whichever type it is). Note that `a` is not assigned a name

- it is already a variable of type `definition` and its members are simply merged with the remaining members of `b`.

You can assign a member of a definition to a variable, using the `[]` or `.` operators. The following assigns `z` to `x` assuming `b` is defined as above :

```
x = b["z"] or x = b.z
```

A member of a definition is simply a variable and hence may be modified by simple assignment; you can also add new members to a pre-existing definition :

```
b.y = 9
b.sqy = b.y ^ b.y
```

`b` now contains `(x:1, y:9, p:"T2", f:data, sqy:81)`.

Note that definitions can hold the arguments of a Metview request, or in other words, Metview requests use definitions as input. Hence, you can use definition type variables to add new or modify existing input elements to a Metview request, e.g. :

```
param_def = (
    param      :    "Z",
    levelist   :    850,
    type       :    "FC",
    date       :    -1,
    step       :    24
)

# retrieve as a LL grid or not according to user choice
if (use_LL = "yes") then
    param_def.grid = [1.5,1.5]
end if

Z_ret = retrieve(param_def)
```

In the above example, a new member named `grid` is added to the `param_def` definition depending on the value of another variable. The resulting definition is then used as input to a MARS retrieval. This can be usefully employed as a means to reduce the amount of written code, e.g. where you have a number of similarly specified calls to `retrieve()` :

```
common_input = (
    levtype    :    "PL",
    levelist   :    850,
    date       :    -1,
    time       :    12,
    grid       :    [2.5,2.5]
)

Uan = retrieve(
    common_input,
    param      :    "U",
    type       :    "AN"
)

Van = retrieve(
```

```

common_input,
param      :    "V",
type       :    "AN"
)

```

Limitations of Definitions

Note that the internal representation of a definition is based on the internal format of Metview requests, as used by icon-functions (see "Icon-Functions" on page -50). Because this data format is missing some of the advanced list-handling features that the macro language has, the following restrictions apply when assigning list members to a definition :

- A list containing other lists is flattened, as long as the depth of the list is no greater than 2; deeper lists cause an error. For example, [1, [2, 3]] becomes [1, 2, 3].
- A single-element list is converted into a scalar. For example, [5] becomes 5.
- An empty list, [], becomes nil.

Fieldsets

A fieldset is an entity representing several meteorological fields, such as the output of a MARS retrieval or the contents of a GRIB file.

The Metview Macro language has a large number of functions available for the input, output, handling and operating on fieldsets - these range from simple arithmetic operations to functions specific for meteorological purposes (e.g. `unipressure()`, `unithickness()`) - see "GRIB input" on page II- 33, "Filtering fieldsets (GRIB data)" on page II- 36, "GRIB output" on page II- 40 and "Functions and Operators on Fieldsets" on page II- 139.

How operators and functions work on fieldsets

Let x_k be the k th field of the fieldset X , and x_{ik} the i th value (grid point or spectral coefficient) of field x_k .

If the two operands are two fieldsets, the operation is carried out between each pair of corresponding field values. The result is another fieldset. Thus :

$$Z = X+Y$$

is equivalent to :

```

for each field k
  for each value i
    Zik = Xik + Yik

```

If one operand is a scalar and the other a fieldset, the operation is carried out between each fieldset value and the scalar. The result is another fieldset. Thus :

$$Z = X+n$$

is equivalent to :

```
for each field k
  for each value i
    Zik = Xik + n
```

The same logic applies to functions. If the argument of a function is a field, the result is a field where each grid point is the result of the function at the corresponding grid point in the input field. Thus :

$$Z = f(X)$$

is equivalent to :

```
for each field k
  for each value i
    Zik = f(Xik)
```

Boolean operators such as > or <= produce 0 when the comparison fails, or 1 if it succeeds. Thus :

$$Z = X > 0$$

gives a fieldset where all the values are either 1 or 0 depending on the corresponding values of the fieldset x being above 0 or not.

NOTE : A fieldset resulting from operations between two fieldsets will have a GRIB header derived from the GRIB header of the first fieldset used in the computations.

Indexing fieldsets

Indexing a fieldset allows you to access particular fields inside it. Indexing uses the square bracket operator []. At its simplest you can use it to extract or refer to a single field inside a fieldset, in the same way as you use the numerical index of an array (a fieldset is just treated as an array of fields).

$x[i]$ = *i*th field of fieldset x:

```
# copies field 2 of fieldset X into Y
Y = X[2]
```

More sophisticated usage of [] allows you to extract or refer to a range of fields with or without the use of an ordinal increment.

$x[i, j]$ = all fields of fieldset x from the *i*th to the *j*th :


```
# copies fields 3, 4, 5, 6, 7 and 8 of X into Y
Y = X[3,8]
```

$X[i, j, k]$ = every k th field of fieldset X , from the i th to the j th :

```
# copies fields 1, 5, 9, 13, 17 of X into Y
Y = X[1,20,4]
```

Indexing of fieldsets can be used to perform some type of calculations in a very efficient manner - the prime example is to derive time step quantities from cumulative fields, e.g. to derive 24h precipitation from the standard cumulative precipitation fields : indexing avoids the use of a loop to carry out the subtraction of consecutive time steps.

The following example shows how to do it :

```
# Retrieve all the timesteps for convective precipitation ("cp") and
# large scale precipitation ("lsp") parameters
r = (
  grid      : [2,2],
  date      : -1,
  type      : "fc",
  levtype   : "sfc"
  step      : [24,"to",240]
)

lsp = retrieve(
  r,
  param     : "lsp"
)

cp = retrieve(
  r,
  param     : "cp"
)

# Compute the total precipitation
total = cp + lsp

# Precipitation is a cumulative field - subtract every two
# consecutive time steps to derive the 24h precipitation amounts
n      = count(total)
rain   = total[1]
rain   = rain & (total[2, n] - total[1, n-1])
```

Merging fieldsets

Merging allows you to create a single fieldset out of several fields or fieldsets. Merging fieldsets is carried out by the concatenation operator (&) or by the `merge()` function. The output of both is a fieldset with as many fields as the total number of fields in the input fieldsets :

```
z = x & y & z & ...
```

or

```
z = merge(x, y, z, ...)
```

Merging with the value `nil` does nothing, which can be used when building a fieldset from scratch, to initialise the merging process :

```
fs = nil
for d = 1997-09-01 to 1997-09-30 do
  x = retrieve(date : d, ...)
  fs = fs & x
end for
```

Notes on fieldset calculations

- When using operators, remember that not all of them can be used with fieldsets in spherical harmonics, e.g. multiplication by a scalar is OK, addition of a scalar is not.
- Functions accepting fieldsets as input may also have restrictions on the spatial representation of their input, i.e. some will only accept Lat/Long fieldsets. This is detailed in their entry on the function reference list (see "Functions and Operators on Fieldsets" on page II-139). In general, functions which require spatial information for the field values will only work with fields in the following representations: regular lat/long, rotated lat/long, reduced lat/long, regular Gaussian, reduced Gaussian, polar stereographic and Lambert. Fields represented in spherical harmonics, satellite projection or in unknown representations may still be worked with, but functions requiring spatial information will fail and issue an error message.
- All computations on fieldsets are done using double precision numbers, but the GRIB encoding/decoding routines use single precision numbers.
- No check is done on the type of data combined in the formulae. It is the user's responsibility to make sure the computations have a physical meaning.
- The resulting fieldsets are encoded into GRIB using the header of the first fields encountered in a computation.

Observations

Due to the nature of the data, the macro language provides little in the way of functions to process observations. Once you retrieve them from MARS or read them from a BUFR file you can only merge and/or plot them (see "Functions and Operators on Observations" on page II- 167). If you need to do more, you have to convert them to geopoints using the `obsfilter()` function - see "Filtering observations (BUFR data)" on page II- 37.

Geopoints



Geopoints is the format used by Metview to handle spatially irregular data (e.g. observations) in a non-BUFR format.

The Metview Macro language has a number of functions available for the input, output, handling and operating on geopoints (e.g. filtering, interpolating and combining with fieldsets or images) - see "Geopoints input" on page II- 34, "Filtering geopoints" on page II- 37, "Geopoints output" on page II- 41 and "Functions and Operators on Geopoints" on page II- 169. Geopoints can be visualised with the `plot()` command and can be customised using symbol visual definitions - see "Symbol" on page III- 217.

Note that if you create a geopoints file by hand-editing an empty text file, Metview will assign a Notes icon to the file and thus some of the interactive usage of the file will be lost. This can be corrected by editing the 'dot file' (see "Icons Revealed" on page I- 32) and restarting Metview; the correct `ICON_CLASS` is `GEOPOINTS`. Alternatively, you can make a copy of an existing geopoints icon from within Metview and start editing that.

Format details

A geopoints file is an ASCII file containing a header section and a data section. The data elements that are present in a geopoints file are the point coordinates (latitude and longitude), the level, date, time and value of either one or two parameters. A two-parameter geopoints file is considered by the plotting engine to contain the components of a vector quantity such as wind. An example of a standard one-parameter geopoints file follows :

```
#GEO
PARAMETER = 12004
lat      long      level   date      time      value
#DATA
36.15    -5.35           0       19970810  1200      300.9
34.58    32.98           0       19970810  1200      301.6
41.97    21.65           0       19970810  1200      299.4
45.03    7.73            0       19970810  1200      294
45.67    9.7             0       19970810  1200      302.2
44.43    9.93           0       19970810  1200      293.4
...
```

The above is a geopoints file containing dry bulb temperature at 2m (`PARAMETER = 12004`). The elements that must be present are the line tagged with the keyword `#GEO`, the line tagged with the keyword `#DATA` and obviously the data points.

Note that a time should be expressed as HHMM; a time of 12 will be interpreted as 0012, ie 00:12.

There are four ways in which data may be specified within a geopoints file. The first is shown above. The remaining three require an additional line in the header section to specify the format as described below. This line must start with `#FORMAT` followed by the name of the format being

used. Optional information about the process that generated the data is possible; currently only #DB_INFO is supported (see below). Anything else between the #GEO and #DATA tagged lines is taken as a comment and skipped - you can use this to add any information required to the file. Such a file is read directly by Macro into a `geopoints` variable.

#FORMAT XYV

This format allows data to be specified with just three columns: X (longitude), Y (latitude) and V (value). The start of an example file would look like the following:

```
#GEO
#FORMAT XYV
PARAMETER = 12004
x/long  y/lat      value
#DATA
-5.35   36.15        300.9
32.98   34.58        301.6
21.65   41.97        299.4
...
```

#FORMAT XY_VECTOR

This format allows two parameters to be stored as the components of a two-dimensional vector (for example uv wind components). The start of an example file would look like the following:

```
#GEO
#FORMAT XY_VECTOR
# lat  lon  height  date      time      u          v
#DATA
80     0    0         20030617  1200     -4.9001   -8.3126
80     5.5  0         20030617  1200     -5.6628   -7.7252
80     11   0         20030617  1200     -6.42549  -7.13829
...
```

#FORMAT POLAR_VECTOR

This format allows two parameters to be stored as the speed and direction of a two-dimensional vector, the direction being specified in degrees where zero is due South and angles increase clockwise. The start of an example file would look like the following:

```
#GEO
#FORMAT POLAR_VECTOR
# lat  lon      height  date      time      speeddirection
#DATA
50.97  6.05      0         20030614  1200      4    90
41.97  21.65     0         20030614  1200      5    330
35.85  14.48     0         20030614  1200      11   170
...
```

As the above file formats indicate, only one or two meteorological parameters may be stored in a `geopoints` file. For `polar_vector` `geopoints`, only the first value (speed) is considered during operations. For `XY` `geopoints`, both values are considered during most operations where it makes sense to do so.

Currently the level, date and time can only be used for filtering; they must be present in the file but you can specify any dummy value if you do not intend to use them.

Storing Data Origin Information in a Geopoints File

It is possible to store details of the origin of the geopoints data in its file. The following example shows the meta-information generated by an ODB query.

```
#GEO
lat      long      level      date      time      value
#DB_INFO
DB_SYSTEM: ODB
DB_COLUMN: lldegrees(lat@hdr);lldegrees(lon@hdr); ; ; ;obsvalue@body
DB_COLUMN_ALIAS: lldegrees(lat);lldegrees(lon); ; ; ;obsvalue@body
DB_PATH: /tmp/cgi/odb/ECMA.conv
DB_QUERY_BEGIN
select lldegrees(lat), lldegrees(lon), obsvalue from hdr, body where
varno=$t2m and obsvalue is not null
DB_QUERY_END
#DATA
36.150  -5.350 0.000  0  0000 295.10000000
```

Currently, ODB is the only database system to generate this meta-information. It is not discussed in detail here because it is generated automatically. Macro functions exist to extract this information from a geopoints variable - see page II-172.

Operations between geopoints and fieldsets or images

When you carry out an operation between geopoints and fieldset (or images) variables the result is another geopoints variable :

- When operating with fieldsets, the values of the field(s) at the geopoints locations are calculated by interpolation and the resulting field values undergo the operation with the geopoints values
- When combining with an image no interpolation takes place; the pixel values where the geopoints are located are extracted and these undergo the operation with the geopoints values
- Unless otherwise stated in the operator or function description, only the first value of a two-valued geopoint is considered during a calculation

Combinations include algebraic operations, boolean operations and a number of functions. See "Functions and Operators on Geopoints" on page II- 169 for details.

Missing values in geopoints

When you combine fieldset data with geopoints, you may end up with some missing values in your geopoints variable. These will have the value contained in the built-in global variable `geo_missing_value` (see "Variables" on page II- 3 for its actual value). Any operation on a geopoints variable will bypass missing values (eg `mean()`) or retain them unaltered (eg `max()`); see individual function descriptions for more details.

In order to remove missing values from a geopoints variable, use the function `remove_missing_values()` as illustrated below:

```
geo_clean = remove_missing_values (geo_source)
```

NetCDF

NetCDF stands for Network Common Data Form. NetCDF is an interface for array-oriented data access and a freely-distributed collection of software libraries for C, Fortran, C++, Java, and perl that provide implementations of the interface. The netCDF libraries define a machine-independent format for representing scientific data. Together, the interface, libraries, and format support the creation, access, and sharing of scientific data. For full details on NetCDF please have a look at <http://www.unidata.ucar.edu/packages/netcdf/>.

NetCDF can be used to handle a great number of scientific data types. Metview uses NetCDF as an internal format for representing data units which can't be conveniently represented by GRIB, BUFR or geopoints. E.g. to handle data arrays representing cross-sections and cross-sectional averages, vertical profiles, Hovmøller matrices, time series (metgrams), etc.. Such NetCDF data files can then be imported by other NetCDF conversant software if users so wish.

Metview provides a set of NetCDF handling macro functions, allowing users to apply a number of functions and operators to NetCDF data. See "NetCDF input" on page II- 34, "NetCDF output" on page II- 41 and "Functions and Operators on NetCDF" on page II- 179.

Deriving NetCDF output in Metview Macro

NetCDF is the format chosen to hold the output of Metview applications, such as Cross-Section, Average and Vertical Profile, which in Macro are represented by the functions `cross_sect()`, `xs_average()`, `vert_prof()`. Such output can't be conveniently represented by the GRIB, BUFR or geopoints format.

A typical example of creating a NetCDF output is :

```
# retrieve some data ...
t = retrieve(
    levelist : [1000,850,700,500,200,100,10],
    param    : "t"
)

# ... derive a cross section - output is netcdf variable
t_xs = cross_sect(
    line           : [20,-30,75,35],
    interpolate_values : "no",
    data           : t
)
```

The output of the `cross_sect()` function is a 2D matrix of data along an arbitrary horizontal coordinate and vertical coordinate (pressure or model levels). Likewise for `xs_average()`. The function `vert_prof()` returns a 1D array of data values along a vertical coordinate (pressure or model levels). In all cases, you can have a number of these data elements held in the same netcdf variable, e.g. if cross-sectioning multi-level fields over several forecast steps, dates or ensemble members.

When using these applications, you can choose to interpolate vertically the data into regular intervals, in which case the result should not be used for subsequent computations, e.g. averages of series of cross-sections.

You can investigate a netcdf file, by extracting its metadata values using the `attributes()` function :

```
attr_list = attributes(netcdf)
```

this returns a definition (see "Definitions" on page II- 17) holding the attributes of the variable, e.g. date, time, levels, etc. :

```
date = attr_list.DATE
levelist = attr_list.LEVELIST
```

How operators and functions work on NetCDF

NetCDF files can contain a wide variety of data. They can contain a number of data units, e.g. a set of cross section plots, a set of 2D geographical grids, a set of time series or vertical profiles, etc.,. Each component of a set is stored in the netcdf file as a separate *netcdf variable*. The handling of and computations with netcdf files are based on the concept of *current variable* :

Out of the N variables contained in a NetCDF file, one is always set to be the current variable. *Functions and operators acting upon the netcdf file will act only upon the current variable.*

In general, users set the current variable (which by default is the first variable in the file) to one of those contained in the file and can then apply a number of functions and operators to them :

```
netcdf3 = netcdf1 op netcdf2

out = function(netcdf1, ...)
```

However, because functions and operators work only on the current variable, when the netcdf contains more than one variable (a multi-variable netcdf), special care must be taken when you need the operator/function to apply to all variables - this is detailed in the next section - "Working with multi-variable NetCDF files" below.

When two netcdf variables are involved, *both files have to have the same number of data points* in each current variable, as an operation between two netcdfs is carried out between each pair of corresponding data values. Thus :

<code>nc3 = nc1+nc2</code>	corresponds to	for each data value i <code>nc3i = nc2i + nc1i</code>
<code>nc2 = nc1+a</code>	corresponds to	for each data value i <code>nc1i = nc1i + a</code>
<code>nc2 = f(nc1)</code>	corresponds to	for each data value i <code>nc2i = f(nc1i)</code>

NOTE : Like fieldsets, a netcdf resulting from an operation on two other netcdfs, will take the meta-data (e.g. date, time, parameter, levels, ...) from the first netcdf.

Working with multi-variable NetCDF files

Functions and operators working on netcdf files, apply only to the current variable. When the netcdf file contains several variables, you need to address each variable separately and explicitly and apply the function/operator to each in turn. For this purpose, Metview Macro provides functions to query the contents of a netcdf file and to set one of its variables to be the current variable :

Users can list the variables held in a netcdf variable by means of the function `variables()` :

```
var_list = variables(netcdf)
```

this returns a list of strings, each holding a variable name. Counting the number of elements in the output list gives you the number of variables

To set one of the existing variables to be the current variable, use function `setcurrent()` :

```
setcurrent(netcdf, n)
```

where `n` is the number of the variable to be set as current.

The two functions above provide the basic framework to operate on multi-variable netcdf files :

Example 1 : To operate on a netcdf file which you want to overwrite with the new results

```
# Derive a cross section of temperature data in a netcdf variable
(...)

# derive the list of netcdf variables
var_list = variables(temp_xs)

# loop over variables and subtract scalar
for i = 1 to count(var_list) do
  setcurrent(temp_xs, i)
  temp_xs = temp_xs - 273.15# acts on current variable only
end for
```

Example 2 : To operate on two netcdf files, assigning the result to a third netcdf, you should create the output netcdf first by a simple copying of one of the input netcdfs :

```
# Derive cross sections of temperature forecast and analysis
# in two netcdf variables, tfc_xs and tan_xs....
(...)

# derive the list of netcdf variables
var_list = variables(tfc_xs)
```



```

# create output netcdf
diff_xs = tfc_xs

# loop over variables and create fc-an difference cross-section
for i = 1 to count(var_list) do
  setcurrent(tan_xs, i)
  setcurrent(tfc_xs, i)
  setcurrent(diff_xs, i)
  diff_xs = tfc_xs - tan_xs
end for

```

Accessing individual NetCDF values

If you need to have access to the individual data values of a netcdf current variable, you can use function `values()` :

```
val_list = values(netcdf)
```

which returns a list with all the values for the current variable. However, you need some way to navigate within this list of numbers. For this purpose, you can find out the dimensions of the data held in a netcdf through the use of the function `dimensions()` :

```
dim_list = dimensions(netcdf)
```

which returns a list of numbers - e.g. if the current variable is a cross section, the output list would have two numbers, the first the number of levels, the second the number of points along the horizontal.

E.g. for the case of a two dimensional variable such as a cross-section, the following code loops over all its data values :

```

dim = dimensions(xsect)
xs_vals = values(xsect)
for j = 1 to dim[1] do
  for i = 1 to dim[2] do
    index = (j-1)*dim[2] + i
    val = xs_vals[index]
  end for
end for

```

An alternative method for accessing individual values is to use the function `value()` :

```
val = value(netcdf, n)
```

which returns the n^{th} value from the current netcdf variable.

Satellite Images

The Metview Macro language provides a limited set of functions applicable to satellite images - see "Functions and Operations on Images" on page II- 163.

An image must be a METEOSAT image, coded in GRIB and representing a grey scale coded using a fixed number of bits per pixel (only 8 bits per pixel are presently supported).

If n is the number of bits used to code a pixel, the macro language functions translate the values from the integral interval $[0 \dots 2^n - 1]$ to the real interval $[0 \dots 1]$, perform the required operation, and translate the result back to the integral interval $[0 \dots 2^n - 1]$. To invert an image, you can write :

```
invert = 1 - image
```

This scheme makes the macro functions independent of the number of coding bits.

Note

Meteosat satellite images archived in MARS are 2500 pixels wide and 2500 pixels high. Each pixel uses one byte, so a satellite image uses 6Mb of memory and disk space. Metview uses memory mapping of files to reduce the amount of resources used when handling images. Nevertheless, remember that a statement like :

```
image3 = max(image1, image2)
```

will process 18 Megabytes.

Arrays - Vectors and Matrices

Vectors and matrices are under development. No applications have yet required the use of vectors or matrices. If there is a demand, more functions will be implemented. In the meantime, all the macro functions use lists of numbers as parameters.

Vectors are created using the `vector()` function and matrices with the `matrix()` function. Their elements are read or set using the `[]` operator :

```
# Allocate a vector of 5 elements
v = vector(5)

# Initialise their value
for i = 1 to 5 do
    v[i] = i
end for
```

Vector literals can be written using the `|` character :

```
v = |3,6,7,9,10|
```

Similarly, matrices are allocated using the `matrix()` function:

```
# Allocate a 5 by 5 matrix
m = matrix(5,5)

# Initialise the matrix to the identity matrix
for i = 1 to 5 do
  for j = 1 to 5 do
    m[i,j] = (i = j)
  end for
end for
```

and literal matrices can be written:

```
m = |1,0,0|
     |0,1,0|
     |0,0,1|
```

INPUT AND OUTPUT IN MACRO

This section deals with file input and output in Macro programs. Input refers to the reading of GRIB files, BUFR files, geopoints files and ASCII files. Output is concerned with writing GRIB files, BUFR files, geopoints files, ASCII files and printing to the console. The functions involved are :

- `read()` function - for all input
- `write()` and `append()` functions - for all output
- `print()` - for all printed output to console

Macro also provides a function `exist()`, to check for existence of input files - this can be used in conjunction with functions which exit a macro, such as `fail()` and `stop()` (see "Inlined or External" in "Macro Run-Modes" on page II- 99), enabling you to handle input errors gracefully.

A list and synopsis of the Metview I/O functions is presented in "File I/O Functions" on page II-187.

Input : `read()` Function

Overview

The input of any file (irrespective of type and format) into Macro is done by the `read()` function. The function can read data in any of the recognised formats :

- GRIB format for model fields and satellite images
- BUFR format for observations
- ASCII matrices of lat-long gridded data
- ASCII lists of irregularly spaced data points (Geopoints)
- NetCDF for other data (e.g. cross sections, vertical profiles)

If the file is not in one of the recognised formats and is ASCII it is also read seamlessly. Binary files in unrecognised formats will cause the function to fail.

The `read()` function takes a string as its single argument, the string being the path and name of the data file. If running in batch mode, a relative path is sufficient (ie the path needn't be specified if the file is in the same folder as the macro program. However, when running the macro interactively, a full path *is* required as Metview will use a temporary current directory. You can specify any file in any reachable directory provided you have suitable permissions.

To use the function you simply assign what the `read()` function returns to a variable. This variable will be of a type corresponding to the file contents. Hence, input of data in recognised formats takes only a trivial single line of code, e.g. to read a set of model fields simply do :

```
tdata = read("/home/xy/xyz/metview/T_grib")
```

The variable `tdata` will be of type fieldset, will hold as many fields as contained in the GRIB file and can be used immediately in whatever computations you require.

To check if a given file exists, use the `exist()` function, which takes a file name as argument. It returns 1 if the file exists and 0 otherwise :

```
if (not(exist("/home/xy/xyz/metview/grib_file"))) then
  fail("specified input file does not exist!")
end if
(...)
```

GRIB input

As outlined above, to input a GRIB file to a macro program use the `read()` function with the file name and path as the single argument :

```
# read a GRIB file
T_data = read("/home/xy/xyz/T_grib")
```

The above assigns the contents of the GRIB file `/home/xy/xyz/T_grib` to a fieldset variable `T_data` which can then be used in any calculations you want. See "Fieldsets" on page II- 19, "Filtering fieldsets (GRIB data)" on page II- 36, "GRIB output" on page II- 40 and "Functions and Operators on Fieldsets" on page II- 139.

ASCII matrix input

If you have (regular gridded) data in ASCII matrix format use `read()` with the file name and path as input.

```
# read a GRIB file
my_matrix = read("/home/xy/xyz/metview/ascii_matrix")
```

In the same way as for GRIB files, Metview assigns the contents to a fieldset (see above for further references).

BUFR input

BUFR file input is carried out as such :

```
# read a BUFR file
my_buf_r = read("/home/xy/xyz/metview/bufr_obs")
```

The above assigns the contents of the BUFR file to a BUFR variable. Note that to perform any significant work with observations data these have to be converted to geopoints, using the `obsfilter()` function - see "Observations" on page II- 22.

Geopoints input

For input of geopoints data use the `read()` function:

```
# read a geopoints file
my_geopts = read("/home/xy/xyz/metview/geopts_file")
```

The above assigns the contents of `geopts_file` to a geopoints variable which can be handled according to requirements. See "Geopoints" on page II- 23, "Filtering geopoints" on page II- 37, "Geopoints output" on page II- 41 and "Functions and Operators on Geopoints" on page II- 169.

NetCDF input

For input of NetCDF data, use the `read()` function :

```
# read a NetCDF file
xsct_netcdf = read("/home/xy/xyz/metview/xs_netcdf_file")
```

The above assigns the contents of `xs_netcdf_file` to a NetCDF variable which can be handled according to requirements. See "NetCDF" on page II- 26, "NetCDF output" on page II- 41 and "Functions and Operators on NetCDF" on page II- 179.

General ASCII input

General ASCII files can be easily input to Macro. Usually the typical ASCII file users import into Metview, contains data values in some regular table/column like arrangement, e.g. comma/space/tab separated values.

To input any ASCII file into Macro simply use the `read()` function :

```
# read an ASCII file
my_text = read("/home/xy/xyz/metview/data.txt")
```

`read()` acting on a text file, reads the whole file in one go and assigns its contents to a list variable. The elements of this list variable are all strings :

Each set of characters up to a carriage return (i.e. each line of text) is one element of this list. If the text file does not end with a carriage return the last character will not be read. Given a text file `my_table` such as :

```
a 5 3 6 2
b 6 3 3 7
```

the following code :

```
x = read("my_table")
```

```
print (x)
print (x[1])
```

will print out :

```
[a 5 3 6 2,b 6 3 3 7,c 0 9 2 6]
a 5 3 6 2
```

Each element of the list `x` is a line of text (e.g. the first element is "a 5 3 6 2") and its type is string. A simple loop in `x` will allow each line to be accessed in turn.

To access each individual element of these lines (i.e. to recover each element of the text file), you can use the `parse()` function : the `parse()` function splits the input string at each occurrence of a user defined string of field separators - the default separator is space (" ") and multiple separators such as comma and space (" , ") are allowed; it returns a list whose elements are the fragments (or "tokens") the input string was split into.

```
x = read("my_table")
f = parse(x[1], " ")
print (x[1])
print (f)
```

The `parse()` function returns a list whose elements are the fragments (or "tokens") the input string was split into; so the above will print :

```
a 5 3 6 2
[a,5,3,6,2]
```

Each element of the list `f` is now accessible by indexing the list. Note also that the type of each list element is correctly determined (i.e. a token which is a number is stored as a number variable), so you can use them in calculations :

```
x = read("my_table")
f = parse(x[2])
print ("line ", f[1], " : s = ", f[5]*f[4]-f[3]*f[2])
```

will print out :

```
line b : s = 3
```

Data Filtering

A topic related to data input is that of data filtering. Filtering consists simply of extracting subsets from a larger data set. Metview Macro provides functions to filter data which has been input through the `read()` function. Some of these filter functions also provide a degree of computational ability, e.g. to convert from one format to another.

When filtering, users specify values for a set of parameters (e.g. date, forecast step, etc.) called *keys* and the sub-sets are formed from the original data by selecting the data for which the values of the keys are identical.

Filtering fieldsets (GRIB data)

To filter fieldset variables (GRIB files) you need to use the macro function which corresponds to the Metview filter icon Data in File. Macro functions which correspond to Metview icons are called **icon-functions** (see "Metview Icon-Functions" on page II- 191 for a full list and synopsis).

The icon-function is named `read()`, so it has the same name as the general input `read()` function. These are however entirely different functions and in usage they can be easily told apart as the icon-functions are the only ones accepting a definition variable (or a definition-like structured set of data) as input.

The icon-function `read()` has a large number of keys you can select for filtering. Have a look at the GRIBFilter icon editor; the most frequently used keys are Parameter, Type, Date and Forecast Step. Additionally, and in the same way as the `retrieve()` icon-function (MARS Retrieval icon), you can operate on the filtered data, though this is restricted to converting between spatial representations (see further below) and/or extracting sub-regions. Example :

```
# Read GRIB file directly, filter a sub-region of Z500
z500 = read(
  source      : "/home/xy/xyz/large_data_grib",
  parameter   : "z",
  level       : 500,
  area        : [75,-20,30,50]
)
```

The above reads the input file and filters and windows data from it. Alternatively, if you have already read the file you can apply the same method to the corresponding fieldset variable :

```
# read GRIB file into fieldset variable
in_data = read("/home/xy/xyz/large_data_grib")

# filter fieldset to get sub-region of Z500
z500 = read(
  data        : in_data,
  parameter   : "z",
  level       : 500,
  area        : [75,-20,30,50]
)
```

Another example of the usage of the icon-function `read()` is to convert between spatial representations. Most fieldsets as retrieved from MARS have their spatial representation as Spherical Harmonics (SH). For some usages, namely as input to some functions - e.g. `integrate()` - you require them in Lat/Long (LL) or Gaussian (GG) grids. If you have SH data and you want to convert it to a LL grid (say), you can use the `read()` icon-function :

```
sh_field = read("/home/xy/xyz/metview//data_in_sh")
ll_field = read(
  grid : [1.5,1.5],
```



```

data : sh_field
)

```

Specifying a lat-long resolution for input element `grid`, converts the data to a LL grid at that resolution.

Filtering observations (BUFR data)

BUFR files contain a number of meteorological parameters (observations) for possibly a number of different dates and times. You may be interested in extracting sub-sets of large BUFR files. For filtering BUFR files you need to read the BUFR file into an observation variable and use the `obsfilter()` icon-function (corresponding to the Observation Filter icon).

Due to the nature of the data, the macro language provides little in the way of functions to process observations. Once you retrieve them from MARS or read them from a BUFR file you can only merge and/or plot them (see "Functions and Operators on Observations" on page II- 167).

For any significant work with observations data these have to be converted to geopoints, also using the `obsfilter()` function - see "Observations" on page II- 22.

The `obsfilter()` function accepts BUFR input, filters out part of that input (optional) and returns the subset as BUFR data (default) or geopoints (if user so specifies). The example below reads data from a BUFR file and filters the 2m air temperature parameter from land synoptic observations into a geopoints variable :

```

buf1 = read("/home/xy/xyz/metview/data_buf1")
my_geopoints = obsfilter(
    output          : "geopoints",
    parameter       : 12004,
    observation_types : "lsd",
    data            : buf1
)

```

Filtering geopoints

Given a geopoints variable you can form subsets according to value, date, level and geographical area, using the macro function `filter()`.

See "Functions and Operators on Geopoints" on page II- 169 for details.

Output: write() and append() Functions

Overview

All output in Metview Macro is handled by the output functions `write()` and `append()`. These functions can output data in any of the recognised formats :

- GRIB format for model fields and satellite images

- BUFR format for observations
- any ASCII information
- Geopoints for irregularly spaced data points (ASCII)
- NetCDF for other data (e.g. cross sections, vertical profiles)

Both `write()` and `append()` take two variables as arguments :

- a file name string or a file handler variable
- the variable holding the data to be output

They perform a similar task in similar ways but as the name implies, `append()` never overwrites pre-existing output while the `write()` function can overwrite or append depending on how you created/opened the output file.

An output file can be created/opened in one of two ways :

- when you first write or append something to it

```
# create an output file by writing something to it
write ("/home/xy/xyz/metview/my_output", output)
```

- when you create a *file handler* variable for your output with the `file()` function

```
# create an output file with a file handler, then write to it
my_output = file("/home/xy/xyz/metview/my_output")
write (my_output, output)
```

Once you have created or opened the output file, you can write to it. You do not have to say anything about the file type or format - Metview determines correctly the file type and format from the type of the variable holding the data to be output. When the macro ends, a suitable icon is automatically assigned to the resulting file.

You can specify the file name with or without the full path. If you do not specify the path, the file is created in the folder where you run the macro.

File handlers or file names?

To output data you either specify the file name everytime you use the output functions or you use a file handler. Using handlers is not just a question of taste or mere convenience, as there are important differences :

- using a file handler enables buffered output implying a slightly different behaviour of the `write()` function in sequential output. Also, if you need to add more output to a *pre-existing* file, you *must* use a file handler
- using a file handler can make it easier to pipe text output into another application (see example in "ASCII output" on page II- 42)
- using a file handler, the output file is not closed until you set the handler variable to zero whereas using a file name string the file is closed as soon as the write or append function has finished

The use of file handlers is therefore advised for all but the simplest of output tasks. In all examples in this manual we have followed this approach.

Writing and appending

You use the `write()` or the `append()` functions depending on whether you are adding new output to a pre-existing file or creating a new file for your output.

Existing output file

If you have an existing file to which you need to add more data, e.g. adding new fields to a GRIB file, adding more data to an ASCII table, you must use the `append()` function, which will place the new data at the end of the file :

```
# retrieve new data
z500 = retrieve(
    parameter : "z",
    level      : 500,
    type       : "an"
)

# append new data to existing file
append("previous_z500_grib", z500)

# this will do the same
gribhan = file("previous_z500_grib")
append(gribhan, z500)
```

If you need to overwrite the contents of the existing file then you must use the `write()` function :

```
# retrieve new data
z500 = retrieve(
    parameter : "z",
    level      : 500,
    type       : "an"
)

# overwrite contents of file with new data
write("current_z500_grib", z500)

# this will do the same
gribhan = file("current_z500_grib")
write(gribhan, z500)
```

New output file

If you are creating a new output file and you need to write sequential output to it, e.g. in a `for` loop, use the `append()` function :

```
# create output file
ff = file("squares")
for i = 1 to 4 do
    square = i*i
    append (ff, i, "square is :", tab, square, newline)
end for
```

This will always add the new output to the existing one. The contents of the output file will be :

```

1 square is :1
2 square is :4
3 square is :9
4 square is :16

```

However, using the `write()` function, what you get depends on whether you are using a file handler - if using a file handler, `write()` does not overwrite the previously written output. So, in the example above it would have made no difference to use either `write()` or `append()` for output.

But if not, then `write()` will overwrite all the previously written output. E.g. :

```

for i = 1 to 4 do
  square = i*i
  write ("squares", i, "square is :", tab, square, newline)
end for

```

Since every call to `write()` clears the previous content, the result of the above piece of code is a text file with a single line :

```

4 square is :16

```

In all, to avoid any confusion, use `append()` to add new input and `write()` to replace existing one.

GRIB output

To create GRIB output (model fields or satellite images) it is enough to pass to the `write()` function a fieldset variable - when this happens a GRIB file is created and a GRIB data file icon assigned to it :

```

# retrieves some fields to fieldset Z_ret...
Z_ret = retrieve(
  levelist : 500,
  param    : "z",
  date     : -1
)

# ...and saves as a GRIB icon named "mygrib"
ff = file("mygrib")
write(ff, Z_ret)
ff = 0

```

BUFR output

To create BUFR output (observations) it is enough to pass to the `write()` function an observations variable - when this happens a BUFR file is created and a BUFR data file icon assigned to it :

```

# retrieves some obs to observations my_obs...
my_obs = retrieve(
  type      : "ob",
  repres    : "bu",
  date     : -3
)

```

```

)

#...and saves as a BUFR icon named "mybufr"
ff = file("mybufr")
write(ff, my_obs)
ff = 0

```

Geopoints output

To create Geopoints output (point data) it is enough to pass to the `write()` function a geopoints variable - when this happens a Geopoints file is created and a Geopoints data file icon assigned to it :

```

# retrieves some obs to observations my_obs...
my_obs = retrieve(
    type      : "ob",
    repres    : "bu",
    date      : -3
)

# converts observations to geopoints
my_pts = obsfilter(
    data      : my_obs,
    output    : "geopoints"
)

# and saves as a geopoints icon named my_geopts
ff = file("my_geopts")
write(ff, my_pts)
ff = 0

```

NetCDF output

NetCDF output from Metview arises only from the use of the application functions `cross_sect()`, `vert_prof()` and `xs_average()`. These functions output data which cannot be represented conveniently by other formats such as GRIB or Geopoints - this need led to the choice of NetCDF for their representation, given its wide user base and existing material for implementation. A typical example of NetCDF data output might be :

```

# retrieves some multi-level fields to fieldset T_ret ...
T_ret = retrieve(
    levelist  : [1000,850,700,500,300,200,50,10]
    param     : "t",
    date      : -1
)

# ... derives the cross-section data ...
xs = cross_sect(
    interpolate_values : "no",
    data               : T_ret
)

# ... and saves as a NetCDF icon named my_xs

```

```
ff = file("my_xs")
write(ff, xs)
ff = 0
```

To create NetCDF output it is enough to pass to the `write()` function a NetCDF variable - when this happens a NetCDF file is created and a NetCDF data file icon assigned to it :

ASCII output

Text output is specified sequentially as arguments to the `write()` and/or `append()` functions. In the same call to the output functions, you can mix text strings, variables (though not dates) and special characters such as line breaks or tabs.

These special characters are specified by means of built-in variables, respectively `newline` and `tab`. The following example shows the usage of the output functions for text :

```
f = file("my_output")

# adds information to the file
write (f, "Results obtained on :", tab)
append (f, yyyyymmdd(date(-1)), newline, newline)

# two sets of numbers - (e.g. could be means from fields)
a = [23,44,55,63,79,12]
b = [87,98,104,123,120,65]
n = count(a)

# output table of their ratio as a percentage
for i = 1 to n do
  ratio = int(100*a[i]/b[i])
  append (f, "ratio ", i, tab, ratio , "%", newline)
end for

append (newline)
append ("finished", newline)
```

This example will create a Notes icon in the folder where the macro is run, with the following contents :

```
Results obtained on :      20010615

ratio 1      26%
ratio 2      44%
ratio 3      52%
ratio 4      51%
ratio 5      65%
ratio 6      18%

finished
```

It is possible to pipe text output into another application from within the macro. The UNIX pipe symbol ("`|`") must be the first character of the output name assigned to the file handler or file name, followed by the name of the program the text is being piped into :

```
function sendmail(recipient,subject,text)
  f = file("|mail " & recipient)
  write(f,"Subject: " & subject)
  write(f,text)
end sendmail

if (exist("zfile.grib")) then
  message = "File found, process continuing..."
else
  message = "File zfile.grb not found. Exiting"
end if

sendmail("you@ecmwf.int","z_prog status",message)
```

ASCII console output

Text output to console is specified exactly in the same format as text output to file, except it uses the `print()` function. The example above provided for ASCII output to file would be similar if you wanted to print to console, just replacing `append()` by `print()` which does not require a file handler or name, e.g.

```
print ("ratio ", i, tab, ratio , "%", newline)
```

The `print()` function outputs to :

- the Metview message window - you can open this by choosing the option Messages in the File menu of the desktop menu bar
- the Macro Editor output area - this is available through the Output icon menu operation or as the Output pane in the Macro icon editor window

You can copy from any of these and paste to a text file.

FUNCTIONS IN MACRO

Macro Function Essentials

Declaring functions

A function is declared in different ways depending on how you specify the list of input parameters (if there is one) :

Usually a function is called with a list of parameters. The function is called only if its number of parameters matches those which are passed to it :

```
function func (p1,p2,...)
  ...
end func
```

If you want to force type checking, the types can be added to the parameter list. The function will be executed only if the number *and* type of the arguments matches those which are passed to it.

```
function func (p1:type,p2:type,...)
  ...
end func
```

A function can be declared without an argument list, in which case it can be called with any number of parameters. However, these have to be retrieved inside the function; for this use the `arguments()` function, which returns a list containing the arguments actually passed to the function :

```
function func      # no arguments given
  loop n in arguments()
    ...
  end loop
end func
```

Finally a function can be declared without arguments:

```
function func() # no arguments allowed
  ...
end func
```


Calling functions and passing arguments

To call a function, you can just use its name and argument list, if the function does not return any values :

```
func(x, 2, 3)
```

If the function returns a value, it must be assigned to a variable in this way :

```
a = func(x, 2, 3)
```

Arguments are passed to functions *by value*, i.e. a copy of the variable is passed to the function and any modifications made to it inside the function are not passed on to the variable outside the function. Hence, *a variable cannot be modified inside a function*.

Scope of functions

The following example demonstrates the scope of functions in Metview. Note that local functions are visible from embedded functions (contrary to variables)

Macro text	Visible functions
extern e1	e1, f1, f3
...	
function f1	
...	e1, f1, f3, f2
function f2	
...	e1, f1, f3, f2
end f2	
...	e1, f1, f3, f2
end f1	
function f3	
extern e2	e1, f1, f3, e2
...	e1, f1, f3, e2
end f3	
...	e1, f1, f3

Function look-up

When a function is called, Macro scans its list of functions according to the following rules:

1 - The currently loaded dictionaries are scanned first (see "Information on icon-function input" on page II- 53).

2 - The functions are searched, starting from any embedded functions up to the global functions. The functions are scanned in the order they are defined. The built-in functions are defined first. Consider the following example :

```

function cos(n)
    return 0
end cos

function a
    print("first a")
end a

function a
    print("second a")
end a

function x

    function cos(n)
        return "cos in x"
    end cos

    function a
        print("third a")
    end a

    function a
        print("fourth a")
    end a

    a()
    print(cos(1.5))

end x

a()
x()
print(cos(1.5))

```

This code will print:

```

first a    # used the first function a() in the main body
third a    # used the first function a() inside x()
cos in x   # used function cos() inside x()
0.0707372 # used built-in function cos(), not user cos()

```

3 - The "fallback" functions are searched. Some functions have been implemented to help writing better code, such as the `count()` function.

4 - The handlers are searched using the same strategy as for the functions.

5 - The directories in the environment variable `METVIEW_MACRO_PATH`, the Macros folder in the Metview folder, and a system wide Macro folder are scanned for a file or icon with the same name as the function :

- If a file is found and determined to be an executable file, the macro loads it as a FORTRAN external (see "Output Formats and Destinations" on page II- 77). Metview determines whether or not a file is executable by searching for the word 'executable' within the string returned by the shell command 'file'.
- If the file exists and is not determined to be executable, it is interpreted as a macro
- If the macro does not contain the wanted function, the next directory is scanned

- When a macro is loaded during this process, all the other functions and global variables defined in the same macro are also loaded.

Recursion

The macro language is fully recursive. The classic example is the function to calculate the factorial of a number :

```
function factorial(n)
  if n <= 1 then
    return 1
  else
    return n*factorial(n-1)
  end if
end factorial
```

Building a Library of Functions

Using macro templates

A convenient way to store functions or other pieces of code (frequently used stuff, useful examples, program skeletons), is to store them as templates. If the code to be stored is in a macro editor window, use the **Make a Template** button from the editor window template button.

This code can then be easily retrieved and inserted at any point in a program you are building by using the option **Use a Template** from the editor window template button.

Using macro inclusion

You can include another macro in your macro program at compile time with the `include` command :

```
include "macroname"
```

The included macro will be executed at the point where the `include` instruction is found. You can specify `macroname` using absolute or relative pathnames (relative pathnames are relative to the position of the macro being compiled) :

```
include "/home/gid/uid/metview/myfuncs/macroname"
include "../myfuncs/macroname"
```

In this way you can place small libraries of functions in macro files, stored in a folder of your choice, ready for inclusion. Note that the `include` command is handled by the macro pre-processor, meaning that variables may not be used as part of the pathname.

Using function look-up

Using the rules of function look-up offers a very easy and convenient way to build a library of functions :

- put the function code in a macro by itself
- give the macro the same name as the function
- place the macro in your `metview/Metview/Macros` or in the system wide Macro folder (ask your Metview installer/developer)

Once this is done the function can be called in any macro just like if it was a built-in function. If you stored it in the system wide macro folder it is accessible to all users.

External Functions

The macro language can call `shell` commands and user defined functions written in `FORTRAN` and `C` (see "Output Formats and Destinations" on page II- 77). These functions/commands are declared in the macro language by :

```
extern name
extern name "command"
extern name(p1:type,p2:type)
extern name(p1:type,p2:type) "command"
```

The parameter list follows the same syntax as functions. The macro uses the function name or the optional UNIX shell command to execute the user function.

If you choose the `extern` construct, remember that unless the optional shell command is given, the name of the function and the shell command are the same. The shell command can be usefully employed to call a debugger as in :

```
extern myfunc "xterm -e dbx /home/xy/macros/myfunc"
```

You can also use the `inline` keyword (see "Strings" on page II- 9)

```
extern name(p1:type,p2:type) "language" inline
... inline code valid for the language...
end inline
```

The "language" can be:

- `shell`, for UNIX shell scripts
- `fortran` or `f77`, for `FORTRAN 77` programs
- `fortran90` or `f90`, for `FORTRAN 90` programs
- `C`, for C programs
- `C++`, for C++ programs

- any shell command. The content of the inline text is first saved in a temporary file which is passed as a parameter to the command.
- any other value defined at your site.

If your inline code requires any external libraries, you can specify these by setting the environment variable `MACRO_EXTRA_LIBS` before starting Metview. For example (in C shell) :

```
setenv MACRO_EXTRA_LIBS "-L/usr/mylibs -lmystatpack"
metview -b macro_that_inlines_fortran_with_mystatpack
```

Setting include directories for 'C' or 'C++' programs requires the setting of the environment variable

```
MACRO_EXTRA_INCS
```

It is also possible to specify additional compiler flags for FORTRAN programs by setting the environment variables

```
MACRO_EXTRA_F77_FLAGS
MACRO_EXTRA_F90_FLAGS
```

A simple example of the use of the `inline` keyword with various embedded languages is shown below.

```
#----- Fortran

extern my_fortran "fortran" inline

    program main
    print*, "Hello from Fortran"
    end

end inline

#----- C

extern my_c "C" inline

#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("%s", " Hello from C");
    exit( 0 );
}

end inline

#----- C++

extern my_cxx "C++" inline

#include <iostream>
```

```

#include <stdlib.h>

int main()
{
    std::cout << " Hello from C++" << std::endl;
    exit( 0 );
}

end inline

#----- shell

extern my_shell "shell" inline

    echo " Hello from a shell script"

end inline

#----- main

my_fortran()
my_c()
my_cxx()
my_shell()

```

The output from running this macro is :

```

Hello from Fortran
Hello from C
Hello from C++
Hello from a shell script

```

See also "Implementation Example" on page II- 88 for a more detailed and practical example of using FORTRAN code within a Metview macro.

Icon-Functions

Metview **icon-functions** are Macro functions which correspond to the Metview icons available on your desktops - executing a MARS Retrieval icon or running the icon-function `retrieve()` results in the same outcome. For each of the icons in your desktop you have a corresponding Macro icon-function. A list of icon-functions implemented in the Macro language, their output type and synopsis is given in "Metview Icon-Functions" on page II- 191. Use `value()` (see below) to check their input definition parameters and values these may take.

The connection between desktop icons and macro icon-functions becomes evident if you drop an executable icon inside a Macro editor. The result is the textual translation of the icon contents into a macro icon-function.

Using icon-functions

Metview icon-functions have their argument list structured as a definition, or in different words, they can accept definition variables as its arguments (see "Definitions" on page II- 17). E.g. :

```
r = retrieve(date : -1, param : ["u", "v"])
```

or more frequently (for clarity only) :

```
r = retrieve(  
    date : -1,  
    param: ["u", "v"]  
)
```

alternatively you may use a definition variable to keep the input parameters, as in :

```
wind = (date: -1, param : ["u", "v"])  
r = retrieve(wind)
```

One important point concerns lists. When editing an icon within Metview, a list is specified by elements separated with slashes. When using an icon-function in Macro though, a list is specified in the usual Macro way, using square brackets and commas, as shown above.

With icon-functions you need only specify the input parameters which are non-default. Any input parameter which is not specified assumes its default value. So, if your requirements match the default settings, you need not provide any input arguments :

```
# Specifies the default contour  
def_cont = pcont()
```

Translating icons to macro icon-functions

When using icon-functions, it happens that the input list is fairly complex or cumbersome to specify by hand. It may also happen that you already have an icon suitable for your objectives, and you do not want to code its contents by hand. In the first case it may be simpler to create an icon with the required input list, using its interactive editor and test it to confirm whether it suits your needs.

Once you have an icon in the desktop you can convert its contents into Macro code, i.e. as the corresponding macro icon-function :

- edit a macro icon
- drop the icon to be translated inside the macro editor

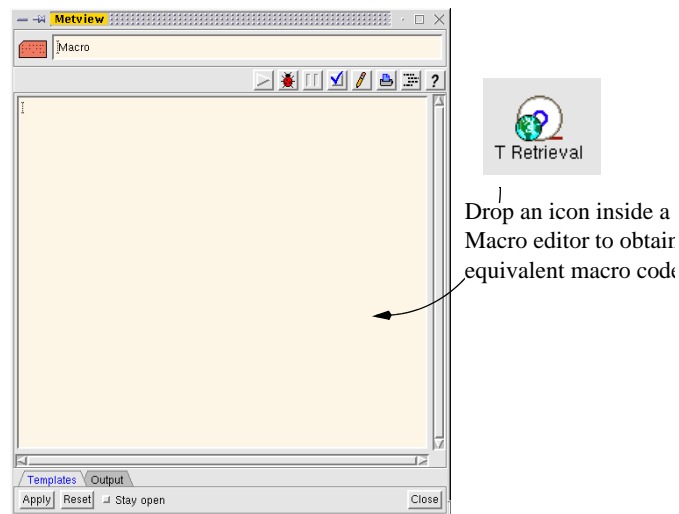


Figure II-2 : To obtain the macro code equivalent to an icon, simply drop it inside a macro editor window.

The code will be inserted at the cursor location, so check where your cursor is before dropping the icon. You can then complement the code at will. You can drop as many icons as you need.

If you need detailed information on any given macro function input, you can use a special macro function called `value()` (see next).

Information on icon-function input

The function `value()` gives you information about icon-functions - e.g. to obtain one of the following two :

- a list of names of all the input parameters that an unction can take
- a list of all the values that one of those input parameters can take

The following examples illustrate the two usages of the `value()` function :

- to obtain a list of all input parameters of an icon-function, you call `value()` with a string - the name of the icon-function - as the single input parameter :

```
# Outputs all parameters of icon-function pcont() to a list
icon = "pcont"
plist = values(icon)

# Writes to a file called "pcont_contents"
filename = icon & "_contents"

for i = 1 to count(plist) do
    write (filename, plist[i],newline)
end for
```

- to obtain a list of all the values that an input parameter of an icon-function can take, you call `value()` with two strings as input arguments - the first being the name of the icon-function, the second the name of the parameter :


```

# Outputs all values that the input parameter "parameter"
# of the icon-function "retrieve" can assume to a list
the_icon = "retrieve"
the_param= "parameter"
plist = values(the_icon,the_param)

# Writes to a file called "retrieve_parameter_values"
filename = the_icon & "_" & the_param & "_values"
for i = 1 to count(plist) do
    write (filename, plist[i],newline)
end for

```

Special Functions

Functions on functions

Macro has three functions that provide some information about other Macro functions :

- `arguments()` - this function returns a list whose members are the arguments with which the function was called. This is useful if you specify a function without an argument list so as to call it with a variable number of arguments. Using the `arguments()` function inside the function allows you to retrieve the arguments for usage. An example was presented in "Declaring functions" above.
- `describe()` returns a description of a function whose name you specify as the input. Bear in mind that this description is a one-line description and documentation of the available functions for this purpose is on-going.
- `dictionary()` returns a list of all the available functions. To write this dictionary to a file you could run the following short macro :

```

# use dictionary to obtain the list of macro functions
dlist = dictionary()

# open a file, dfile is a file handler
filename = "current_dictionary"
dfile = file(filename)

# use a loop to write the list of functions to a file
for i = 1 to count(dlist) do
    write (dfile, dlist[i], newline)
end for

# close the file by setting file handler to 0
dfile = 0

```

Handlers

Handlers are special functions that are called in case of certain events.

```

on name
    ...

```

end name

A parameter list can also be specified using the syntax described for the functions. The only handlers supported at present are the ones used in conjunction with the Metview run modes and user designed graphical interface.

Handlers allow you to code different outcomes for the different ways in which a macro program can be called - see "Macro Run-Modes" on page II- 99 and "Using Macro Parameters" on page II- 110 for details.

VISUALISATION IN MACRO

Overview

The components of a Metview visualisation (see Figure II-3 and the elements provided in "Visualisation - an Overview" on page I- 62) are :

- **Display Window** - which defines the plot arrangement (through its layout) and the plot type and characteristics (through the views). In Metview Macro the display window is specified as the output of the Metview icon-function `plot_superpage()`.
- **Data** - of which there can be several units (model fields, observations,...) simultaneously. In Metview Macro, data units are held as data variables - fieldset, observations, geopotents, NetCDF - resulting from an input operation or some filtering/computation.
- **Visual Definitions** - which control the plotting of the data (contour lines, symbols, observations, titles, axis, ...). In Metview Macro, these are specified as the output of Metview icon-functions such `pcont()`, `psymb()`, `pobs()`, `pcontext()`, etc.,.

A Display Window divided in three Layout Frames, each using a different View :
 a Cross Section View
 a Vertical Profile View
 a Map View in a 3x1 arrangement

Different plot types are generated from the same data unit, according to the views specified by the user

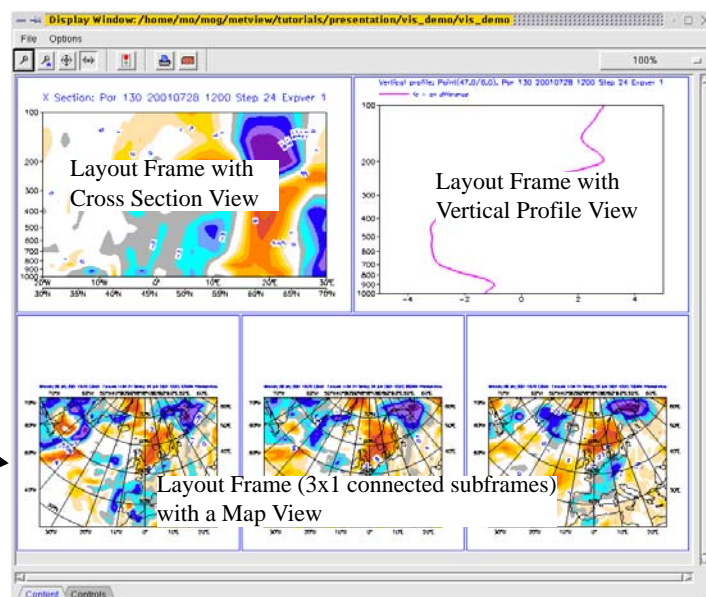


Figure II-3 : An example visualisation. A Display Window is implemented with an arrangement of layout frames (subdivisions) and a different view assigned to each. Different plots can then be produced from the same data unit. Each layout frame can use its own visual definition.

In a few words, data is visualised in a display window, using visual definitions. Under this simple statement, complex visualisations can be obtained, e.g. a multi-plot layout displaying several data units each using different visual definitions. All the visualisation work is handled by the Metview `plot()` function (see next section).

In Metview, you can produce a visualisation in several **formats** :

- PostScript file
- PNG or JPEG graphics file
- On screen display

And send these outputs to several **destinations** :

- a preview program (ghostscript for PS, xv for JPG)
- a printer
- a file on disk

In Macro both format and destination are specified through the function `output()` independently of the `plot()` function so you can use the same plotting instructions to produce output of a different nature.

For a list and synopsis of the Metview functions involved in the visualisation process, please see "Metview Icon-Functions" on page II- 191.

The `plot()` Function

Visualisation in Metview Macro is carried out through the `plot()` function. The `plot()` function takes a variable number of input arguments, supplied in a precise order :

- a **display window** specification provided as a definition variable obtained from the Metview icon-function `plot_superpage()` (corresponds to the Display Window icon on your desktop). This is optional, but in its absence a default specification is used
- a **data** variable - this can be a fieldset, observations, geopoints, NetCDF, containing the data to be visualised
- **visual definition** specifications provided as definition variables obtained from the output of Metview icon-functions such `pcont()`, `psymb()`, `pobs()`, `pcont()`, etc.. These are optional but in their absence defaults suitable for the data in question are automatically provided

You can have any number of (*data, visual definition*) sets. If more than one of these sets is present they are overlaid in the visualisation, thus enabling users to produce plots of several meteorological fields with observations or other suitable point data overlaid. Schematically, the `plot()` function is used as such :

```
plot(dw, data1, visdef1a, visdef1b, data2, visdef2, ...)
```

The above specifies that the variable `data1` is plotted with the visual definitions `visdef1a` and `visdef1b` and the variable `data2` is plotted with the visual definition `visdef2`, in the display_window `dw`. Note that the visual definition(s) to be applied to a given data unit, must follow immediately after the data variable in the call to `plot()`.

The visualisation outcome of the `plot()` function depends on the order in which you specify the input arguments.

Suppose you have the following variables :

`z_500` - a fieldset variable holding a Z field and 500 hPa

`Z_cont` - a definition variable holding a contour suitable for Z data
`Z_text` - a definition variable holding a title suitable for Z data
`T-500` - a fieldset variable holding a T field and 500 hPa
`T_cont` - a definition variable holding a contour suitable for T data
`T_text` - a definition variable holding a title suitable for T data

Given below are a variety of uses of the `plot()` function with the above variables :

- `plot(Z_500)` - Plots `Z_500` in a default display window with default contours and default text
- `plot(Z_500,Z_cont)` - as above but using specified contours
- `plot(Z_500,Z_cont,Z_text)` - as above but using specified text.
- `plot(Z_500,T_500,T_cont)` - Plots `Z_500` with default contours and `T_500` with `T_cont` contours; this shows that the visual definition to be applied to a data must follow it immediately in the argument list of `plot()`.
- `plot(Z_500,T_500,Z_cont,T_cont)` - Plots `Z_500` with default contours and `T_500` with `Z_cont` and `T_cont`.
- `plot(Z_500,Z_cont,T_500,T_cont)` - Plots each field with its own contour.
- `plot(Z_500,Z_cont,Z_txt,T_500,T_cont,T_txt)` - Plots each field with its own contour and title

All the above examples used the default display window as none was specified. If you had defined a display window held in the variable `display` and wanted to use it for the visualisations instead of the default one, you had to specify it as the very first argument to `plot()` :

```
plot(display,Z_500, Z_cont)
```

The outcome of the usage of `plot()` with `display` in all of the previous examples is the same, except that a custom display window would be used.

Finally, two calls to the `plot()` function, without specifying a display window :

```
plot(Z_500,Z_cont)
plot(T_500,T_cont)
```

produce *two separate* default display windows, each with its own field. However, if you specify a display window in both calls to `plot()` :

```
plot(display,Z_500,Z_cont)
plot(display,T_500,T_cont)
```

this produces the same output as a single call with both data units, e.g. :

```
plot(display,Z_500,Z_cont,T_500,T_cont)
```

Overview of Visualisation Layout in Macro

Visualisation layout components

The fundamental component of a visualisation is the display window. Macro provides a way to code any display window layout you need. Of course if your requirements are satisfied by the default display window using the default view (map view), you do not need to specify one. For all other layouts you need to specify the visualisation layout and the views (which together build the display window).

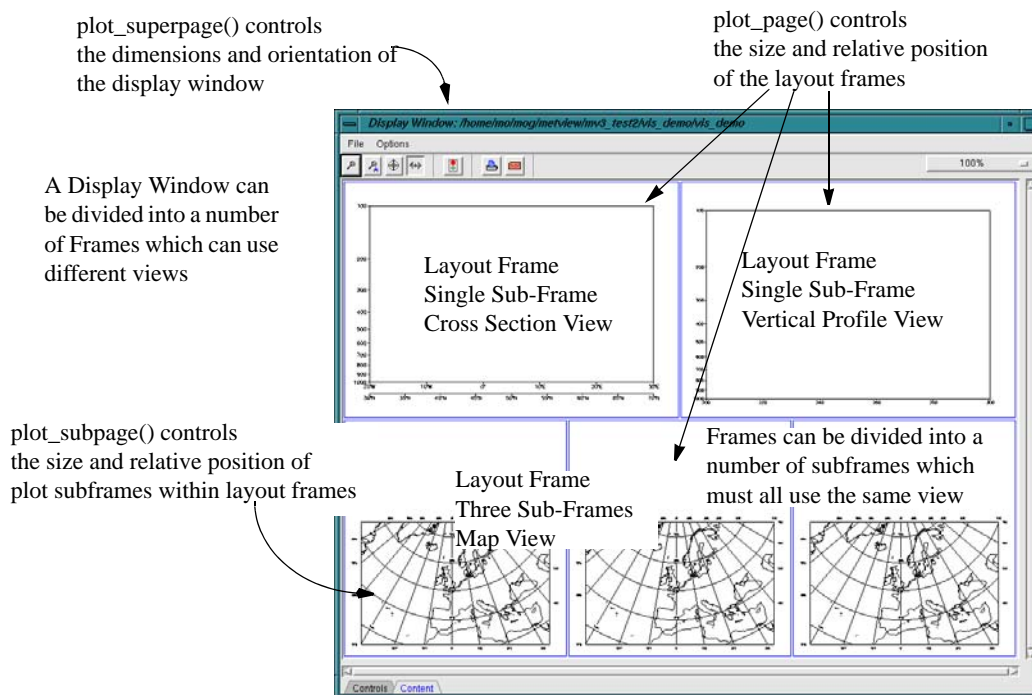


Figure II-4 : An example layout with its components highlighted. The macro functions which control each component are also pointed out - refer to text for details.

Whereas in interactive work you build a display layout through the graphical Display Window icon editor ("The Display Window Icon Editor" on page I- 72), in Macro you use a number of functions to specify the same layout. It is important therefore that you become familiar with the Macro layout entities as described in Figure II-4 . These are :

- The display window
- The layout frame and associated view
- The plot subframe

A display window can have a number of layout frames, each of which can have a different view; the layout frames can be divided in a number of plot subframes. So what is the functional difference between frames and subframes? What would be the difference between a display window with a layout of 2x2 frames and a display window with a layout of one single frame, divided into a 2x2 subframe arrangement?

A practical example makes the difference clear. Suppose you have one data unit (MARS Retrieval, GRIB file) with N fields (e.g. N forecast steps) to be plotted as maps in a 2x2 arrangement.

- If you use a display window with a layout of 2x2 frames in the instruction `plot(dw, data)` : all the N fields will be plotted in the same frame - this is because layout frames are insulated or data-tight, i.e. *data cannot flow from one frame to the other*. This must be so for frames to be able to use different views.
- If you use a display window with a layout of one single frame, divided into a 2x2 subframe arrangement in the instruction `plot(dw, data)` : the N fields will be plotted 4 at a time in the 2x2 arrangement - this is because you are within the same single layout frame and data can flow through the plot subframes. If a layout frame has M plot subframes, a data unit will display M fields simultaneously.

In a nutshell, the number and arrangement of subframes dictates the number and arrangement of plots which are simultaneously visible in a layout frame. The above example is exemplified in Figure II-5.

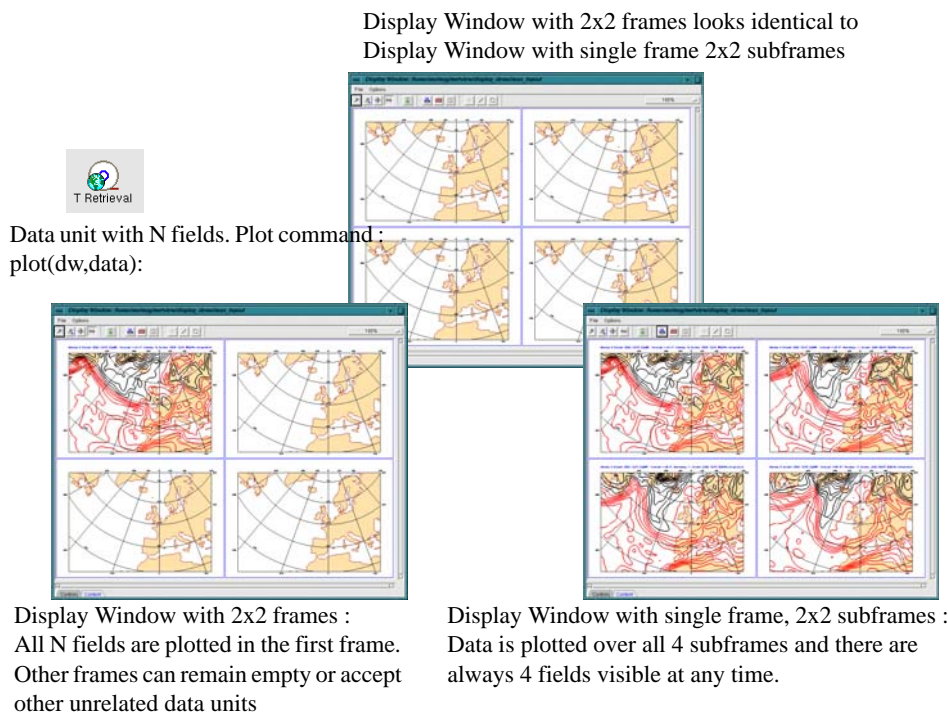


Figure II-5 : Differences in visualisation outcome arising from functional differences between layout frames and plot subframes.

Macro layout implementation

Here we see how the layout components described in the previous sub-section are implemented in the Metview Macro language. MAGICs users will find the nomenclature similar - this is unsurprising since it is MAGICs which takes care of the plotting in Metview :

- The **display window** is known in Macro as the **superpage** - it is specified in the Macro language by the `plot_superpage()` icon-function : this controls the dimensions and orientation of the display window on screen or of the plot on a sheet of paper and corresponds to the Display Window icon on your desktop

- The **layout frames** are known in Macro as **pages** - specified in the Macro language by the `plot_page()` function : it controls the relative dimensions and positioning (i.e. the layout) of the layout frames relative to the display window
- The **plot subframes** are known in Macro as **subpages** - specified in the Macro language by the `plot_subpage()` function : it controls the relative dimensions and positioning (i.e. the layout) of the plot subframes relative to the layout frame to which they belong

These three functions work in a hierarchical way in that the output of `plot_subpage()` (a definition variable holding the *subframe layout*), is input to `plot_page()` whose output (a definition variable holding the *frame layout*) is in turn input to `plot_superpage()`. This last function returns a definition variable holding the display dimension and orientation as well as the layout, which is used as the first argument to the `plot()` function.

It is this hierarchical structure which translates the functional differences between layouts of MxN frames and single frame layouts with MxN subframes.

The final component of a display window are the **views**. These are specified by a number of icon-functions each corresponding to one of the available views - Map, Cross-Section, Vertical Profile, Average, Tephigram, Curve. Since these are attached to each layout frame, they are input to the `plot_page()` function.

The following sections provide a detailed explanation of the visualisation components and how to specify them.

Display Window Specification

A Display Window is specified by the output of the `plot_superpage()` function. This output is a definition (a list of named variables) holding the dimensions and orientation of the display window as well as the frames layout. The input arguments to the `plot_superpage()` function must be specified as a list of named variables as such :

- `layout_size` - a string variable, specifying the display window size type; it can be set to "a4", "a3" - for standard paper sizes - or "custom" - for arbitrary dimensions. Default is "a4"
- `layout_orientation` - a string variable specifying the display window orientation; it can be set to "portrait" or "landscape" but should be used only if `layout_size` is set to one of the standard paper sizes, otherwise it is ignored. Default is "landscape"
- `custom_width` - a number variable defining the width of the display window in cm - if used it overrides the two other variables
- `custom_height` - a number variable defining the height of the display window in cm - if used it overrides the two other variables
- `pages` - a list variable with as many elements as there are frames (pages) in the display window; each of these frames is specified by a call to the `plot_page()` function (see next section)

A few examples follow next.

To obtain an A3 landscape display window divided in 3 frames :

```
display = plot_superpage(
```



```

layout_size      : "a3",
layout_orientation : landscape",
pages           : [page_tleft, page_tright, page_bottom]
)

```

The list of pages input to the `pages` variable defines the layout. From the code above you can only see it specifies a display window divided in three frames; it could be a 3x1, a 1x3 or any arbitrary 3 frame arrangement (even overlapping) and it could have any view attached. You would need to inspect the `page_tleft`, `page_tright` and `page_bottom` variables to find out what the layout exactly is. As it happens this is the specification for the display window shown in Figure II-4.

To obtain a custom size display window, divided in four frames :

```

display = plot_superpage(
    custom_width   : 25,
    custom_height  : 25,
    pages         : [page1,page2,page3,page4]
)

```

Again, the four pages can have any size or relative arrangement.

To obtain an A4 landscape with a single frame with a single subframe (default display window) :

```

display = plot_superpage(
)

```

i.e. as per usual Metview Macro practice, if you need the default display window you needn't use any arguments.

Next, we detail how to specify the layout of the frames, through the function `plot_page()`. In a nutshell, a layout composed of `n` frames must be built with `n` calls to `plot_page()` and the output of each of these calls organised as elements of a list variable which is then supplied to the `pages` parameter in `plot_superpage()`.

Layout Frame Specification

To build a layout composed of `n` layout frames you must call `plot_page()` function `n` times. Each call to `plot_page()` will specify one of the frames in turn, by returning a definition variable holding the coordinates of the frame (expressed as percentages of the display window size) within the display window. Once the `n` layout frames are defined you simply pass them as a list to the `pages` parameter in the call to `plot_superpage()`. The input arguments to the `plot_page()` function must be specified as a list of named variables as such :

- `top` - a number variable specifying the position of the top side of a rectangular frame as a percentage of the display window height; 0 is topmost, 100 is lowest, default value is 0
- `bottom` - a number variable specifying the position of the bottom side of a rectangular frame as a percentage of the display window height; 0 is topmost, 100 is lowest, default value is 100

- `left` - a number variable specifying the position of the left side of a rectangular frame as a percentage of the display window width; 0 is leftmost, 100 is rightmost, default is 0
- `right` - a number variable specifying the position of the right side of a rectangular frame as a percentage of the display window width; 0 is leftmost, 100 is rightmost, default is 100
- `view` - a definition variable which specifies the view to be used in the page; this is supplied as the output of one of the Metview icon-functions corresponding to the available views, e.g. Map, Cross-Section, Vertical Profile, Average, Tephigram, Curve. See "View Specification" on page II- 64 for view specification. The default (which applies if this variable is not specified) is a Map View using the full globe in cylindrical projection
- `subpages` - a list variable with as many elements as there are subframes (subpages) in the frame (page); each of these subframes is specified by a call to the `plot_subpage()` function (see next section). The default (which applies if this variable is not specified) is a single subframe with the same dimensions as the frame (page) itself

A few examples follow - a display window with a 3 frames layout, one along the bottom half of the display using a map view and split in 3x1 subframes and the other two dividing the top half, the left one using a cross-section view, the right one a vertical profile view:

```

page_bottom = plot_page(
    top      :    50,
    bottom   :   100,
    left     :    0,
    right    :   100,
    view     :   a_map_view,
    sub_pages : [subpage1, subpage2, subpage3])

page_tleft = plot_page(
    top      :    0,
    bottom   :   50,
    left     :    0,
    right    :   50,
    view     :   a_xs_view)

page_tright = plot_page(
    top      :    0,
    bottom   :   50,
    left     :   50,
    right    :  100,
    view     :   a_vp_view)

display = plot_superpage(
    layout_size      : "a3",
    layout_orientation : "landscape",
    pages            : [page_tleft, page_tright, page_bottom]
)

```

The views would be supplied by calls to the functions `mapview()`, `xsectview()` and `vert-profview()`. The list of subpages input to the `sub_pages` variable in `page_right` defines the subdivision of this layout frame. The code does not give any indication as to the arrangement of the plot subframes within the layout frame - you'd need to inspect the subpage variables to see what the subframe layout was.

The absence of the `sub_pages` variable from the `page_tleft` and `page_tright` specification implies you will use its default setting - a single subframe with the same dimensions as the frame, i.e. `page_left` and `page_right` will appear as an undivided frame.

Usually, most layouts are regular, i.e. frames are subdivided in a $M \times N$ arrangement. But you can have any irregular arrangement even with overlapping frames, it is simply a question of specifying suitable numbers for the top/left/bottom/right parameters. E.g. a display window with two overlapping frames :

```

page1 = plot_page(
    left :    10,
    right :   60
)

page2 = plot_page(
    left :    40,
    right :   85
)

dw = plot_superpage(
    pages :   [page1,page2]
)

```

Here we omitted the `top` and `bottom` variables so they will take up their default values (0 and 100 respectively).

Plot Subframe Specification

To subdivide a layout frame into n subframes, you need to call the `plot_subpage()` function n times. Each call to `plot_subpage()` will specify one of the subframes in turn, by returning a definition variable holding the coordinates of the subframe (expressed as percentages of the frame size) within the layout frame. Once the n plot subframes are defined, you simply pass them as a list to the `sub_pages` parameter in the call to `plot_page()`.

The input arguments to the `plot_subpage()` function must be specified as a list of named variables as such :

- `top` - a number variable specifying the position of the top side of a rectangular subframe as a percentage of the frame height; 0 is topmost, 100 is lowest, default value is 0
- `bottom` - a number variable specifying the position of the bottom side of a rectangular subframe as a percentage of the frame height; 0 is topmost, 100 is lowest, default value is 100; clearly you must supply a number larger than the one supplied for `top`
- `left` - a number variable specifying the position of the left side of a rectangular subframe as a percentage of the frame width; 0 is leftmost, 100 is rightmost, default is 0
- `right` - a number variable specifying the position of the right side of a rectangular subframe as a percentage of the frame width; 0 is leftmost, 100 is rightmost, default is 100

So to subdivide a page simply define the required number of subpages with suitable coordinates. E.g. a plain 3x1 subdivision of a frame would be specified as :

```

subpage1 = plot_subpage(
    top      :    0,
    bottom   :   100,
    left     :    0,

```

```

        right      :    33.333
    )

    subpage2 = plot_subpage(
        top        :    0,
        bottom     :    100,
        left       :    33.333,
        right      :    66.666
    )

    subpage3 = plot_subpage(
        top        :    00,
        bottom     :    100,
        left       :    66.666,
        right      :    100
    )

    page_bottom = plot_page(
        top        :    50,
        bottom     :    100,
        left       :    0,
        right      :    100,
        view       :    map_euro,
        sub_pages  : [subpage1, subpage2, subpage3]
    )

    # The remaining page and display window definitions
    page_tleft = plot_page(
        top        :    0,
        bottom     :    50,
        left       :    0,
        right      :    50,
        view       :    xs_euro
    )

    page_tright = plot_page(
        top        :    0,
        bottom     :    50,
        left       :    50,
        right      :    100,
        view       :    vp_euro
    )

    display = plot_superpage(
        layout_size      : "a3",
        layout_orientation : "landscape",
        pages             : [page_tleft, page_tright, page_bottom]
    )

```

The code above shows you the specification in Metview Macro of the layout presented in Figure II-4.

View Specification

A View is a plotting specification which controls :

- the type of plot to produce from the data unit - map, cross-section, vertical profile, curve, etc,
- the geographical elements of the plot - this includes projection and area coordinates, transect line, averaging area or point coordinates; also axis limits and specification, geographical grid spacing, land and sea shades.
- the rules that control overlay of multiple data units - controlled by a set of matching switches
- the plot positioning within the subframe
- the plot border specification - on / off plus colour, format and thickness

Each type of plot you can produce in Metview has an associated view. If there is no view for a particular type of plot, this plot cannot be produced in Metview. Since the view determines the plot type, you can produce several types of plot from the same data unit, provided it contains suitable data.

See "The View Concept in Visualisation" on page I- 65 and "Role of Views in Visualisation" on page I- 86 for full details on Views, namely the meaning of their input elements. Here we cover the implementation of Views in Metview Macro.

Views are specified by the icon-functions that correspond to the available views :

- Map View - implemented by `mapview()`
- Cross-Section - implemented by `xsectview()`
- Vertical Profile - implemented by `vertprofview()`
- Average - implemented by `averageview()`
- Tephigram - implemented by `tephigramview()`
- Curve - implemented by `curveview()`

Views are associated to the layout frames hence the view definitions returned by the view icon-functions are input to the `plot_page()` function as shown in the examples above. The display window shown in Figure II-4 uses three different views - a map view, a cross-section view and a vertical profile view. They provide useful examples of view specification in Macro :

The map view - note a separate specification of a coastline visual definition which is used as input to the `coastlines` input parameter; this is the Macro equivalent of the Map View icon accepting a Coastlines icon as input :

```
landsh = pcoast(
  map_coastline_colour      : "black",
  map_grid_colour          : "black",
  map_grid_longitude_increment : 10,
  map_label_colour         : "black",
  map_coastline_land_shade   : "on",
  map_coastline_land_shade_colour : "cream"
)

map_euro = mapview(
  map_projection : "polar_stereographic",
  area           : [24.0, -33.0, 60.0, 65.0],
  coastlines     : landsh
)
```

The vertical profile view - note a separate specification of two axis visual definitions which are used as input to the `pressure_axis` and `value_axis` input parameters; this is the Macro equivalent of the Vertical Profile View icon accepting Axis icons as input :

```
paxis = paxis(
    axis_type      : "logarithmic",
    axis_min_value : 1000,
    axis_max_value : 100,
    axis_orientation: "vertical",
    axis_position  : "left"
)

axval = paxis(
    axis_min_value : -5,
    axis_max_value : 5
)

vp_euro = vertprofview(
    point          : [47,6],
    bottom_pressure : 1000,
    top_pressure   : 100,
    pressure_level_axis: "log",
    pressure_axis   : paxis,
    value_axis      : axval
)
```

The cross-section view - no separate specifications, as the corresponding icon does not accept take any icon as input :

```
xs_euro = xsectview(
    line          : [30,-20,70,30],
    bottom_pressure : 1000,
    top_pressure   : 100,
    pressure_level_axis : "log"
)
```

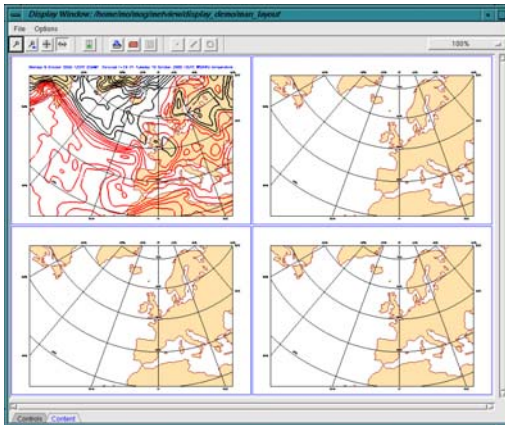
The views are associated with the layout frames; in Macro this translates as the view definitions being used as input to the `plot_page()` function (view parameter) as shown in the code listing presented in the previous two sections.

Views accept other input parameters - please have a look at the icon editors to see a listing of the input for each view icon. Their specification follows the syntax as exemplified above. Remember you can always drop an icon inside a Macro editor to obtain the macro language translation of the icon.

Fast Layout Specification in Macro

It is clear from the previous sections on layout specification in Macro, that even moderately complex visualisations already require a fair amount of coding. Additionally, defining the dimensions of layout elements demands a frequently tedious calculations of proportions and corresponding percentages.

However, there are ways in which you can minimise this workload. For complex layouts or those with a large number of frames or subframes (e.g. EPS "postage stamp" layouts) these are indeed the recommended options. Common sense should make you use them soon after you start using the macro language.



Display Window with 2x2 frames with a data unit dropped in the top left frame

Display Window with single frame, 2x2 subframes with a data unit dropped in the top left frame:

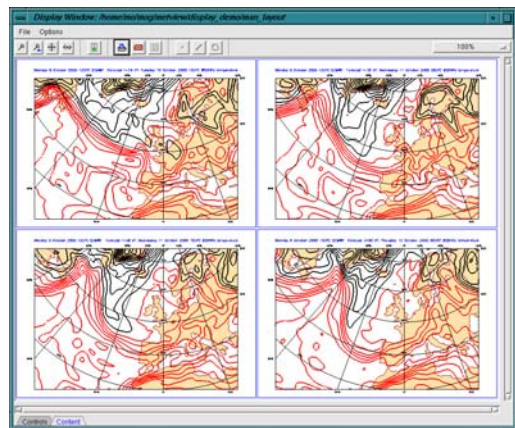


Figure II-6 : Example of two regular layouts one with 2x2 independent frames (top left) another with a single frame split into 2x2 connected subframes. These are used to exemplify the automatic layout functions (see text for details).

A macro function for layouts of MxN frames

Regular layouts of MxN frames all with identical views present a regularity of coding that allows them to be coded automatically by a short macro function. A function for these purposes called `mxn_frames()` is listed below :

```
function mxn_frames (a_view, frame_cols, frame_rows)

delta_x = 100 / frame_cols
delta_y = 100 / frame_rows
frame_list = []
for i = 1 to frame_rows do
  hi_pos= delta_y * (i-1)
  lo_pos = delta_y * i
  for j = 1 to frame_cols do
    left_pos = delta_x * (j-1)
    right_pos = delta_x * j
    frame = plot_page(
      top      : hi_pos,
```

```

        bottom : lo_pos,
        left   : left_pos,
        right  : right_pos,
        view   : a_view
    )
    frame_list = frame_list & [frame]
  end for
end for
return frame_list
end mxn_frames

```

This function takes as input a view definition and two numbers, the first the number of frames along the width of the display window (columns), the second the number of frames along the height of the display window (rows). It returns a list of pages which can be supplied as input to the `pages` variable of the `plot_superpage()` function, as such :

```

map_euro = mapview(
  map_projection : "polar_stereographic",
  area          : [24.0,-33.0,60.0,65.0]
)

page_list = mxn_frames(map_euro, 2, 2)

display = plot_superpage(
  layout_size       : "a3",
  layout_orientation : "landscape",
  pages             : page_list
)

```

So you only need to specify the view to be used and call the function with the required number of columns and rows. The result of the above piece of code is the display window shown in the top left of Figure II-6 above (2x2 frames each with a single subframe).

A macro function for layouts of MxN subframes

A nearly identical function can be written to create layouts of a *single frame* subdivided into MxN subframes. A function for these purposes called `mxn_subframes()` is listed below :

```

function mxn_subframes (subframe_cols, subframe_rows)

  subdelta_x = 100 / subframe_cols
  subdelta_y = 100 / subframe_rows
  subframe_list = []
  for i = 1 to subframe_rows do
    hi_pos = subdelta_y * (i-1)
    lo_pos = subdelta_y * i
    for j = 1 to subframe_cols do
      left_pos = subdelta_x * (j-1)
      right_pos = subdelta_x * j
      subframe = plot_subpage(
        top : hi_pos,
        bottom : lo_pos,
        left : left_pos,
        right : right_pos
      )
    end for
  end for
  subframe_list = subframe_list & [subframe]
end function

```



```

    )
    subframe_list = subframe_list & [subframe]
  end for

  end for
  return subframe_list
end mxn_subframes

```

This function takes as input two numbers, the first the number of subframes along the width of the frame (columns), the second the number of subframes along the height of the frame (rows). It returns a list of subpages already prepared which can be supplied as input to the `subpages` variable of the `plot_page()` function, as such :

```

subpage_list = mxn_subframes(2, 2)

one_page = plot_page(
  view      : map_euro,
  sub_pages : subpage_list
)

display = plot_superpage(
  layout_size      : "a3",
  layout_orientation : "landscape",
  pages            : [one_page]
)

```

So you only need to specify the required number of columns and rows to derive a frame subdivided into $m*n$ subframes. The result of the above piece of code is the display window shown in the bottom right of Figure II-6 (single frame with 2x2 subframes).

A macro function for general regular layouts

Combining the two functions above allows you to use a function which can create a $M*N$ layout of frames, each subdivided in $P*Q$ subframes.

```

function general_layout (a_view, n_cols, n_rows, n_subcols, n_subrows)

  delta_x = 100 / n_cols
  delta_y = 100 / n_rows

  subframe_list = mxn_subframes(n_subcols, n_subrows)

  frame_list = []
  for i = 1 to n_rows do
    hi_pos= delta_y * (i-1)
    lo_pos = delta_y * i
    for j = 1 to n_cols do
      left_pos  = delta_x * (j-1)
      right_pos = delta_x * j
      frame     = plot_page(
        top      : hi_pos,
        bottom   : lo_pos,
        left     : left_pos,
        right    : right_pos,

```

```

                sub_pages: subframe_list,
                view      : a_view
            )
            frame_list = frame_list & [frame]
        end for
    end for

    return frame_list

end general_layout

```

This function takes as input a view definition and four numbers, the first two are the number of frames along the width (columns) and height (rows) of the display window, the second two the number of subframes along the width and height of each layout frame.

It returns a list of pages which can be supplied as input to the `pages` variable of the `plot_superpage()` function, as such :

```

map_euro = mapview(
    map_projection : "polar_stereographic",
    area          : [24.0,-33.0,60.0,65.0]
)

page_list = general_layout(map_euro, 2, 1, 1, 2)

display = plot_superpage(
    layout_size       : "a3",
    layout_orientation : "landscape",
    pages             : page_list
)

```

This piece of code creates a display window divided in two layout frames (left and right), each of which is in turn subdivided in two subframes (upper and lower).

Note that this function combines the functionality of the two other function presented previously. The two layouts presented in Figure II-6, can be simply specified by this function :

The layout on the top left of the figure :

```

page_list = general_layout(map_euro, 2, 2, 1, 1)
display = plot_superpage(
    pages      : page_list
)

```

The layout on the right of the figure :

```

page_list = general_layout(map_euro, 1, 1, 2, 2)
display = plot_superpage(
    pages      : page_list
)

```

Using the Display Window editor

By far the most convenient way to prepare a visualisation layout is simply to use the editor of the Display Window icon to build the required layout. The idea is that you use its interactive graphical interface to design the layout and then convert the result to macro code. This is the recommended course for all non MxN layouts : **use the Display Window editor as a visual layout programming tool.**

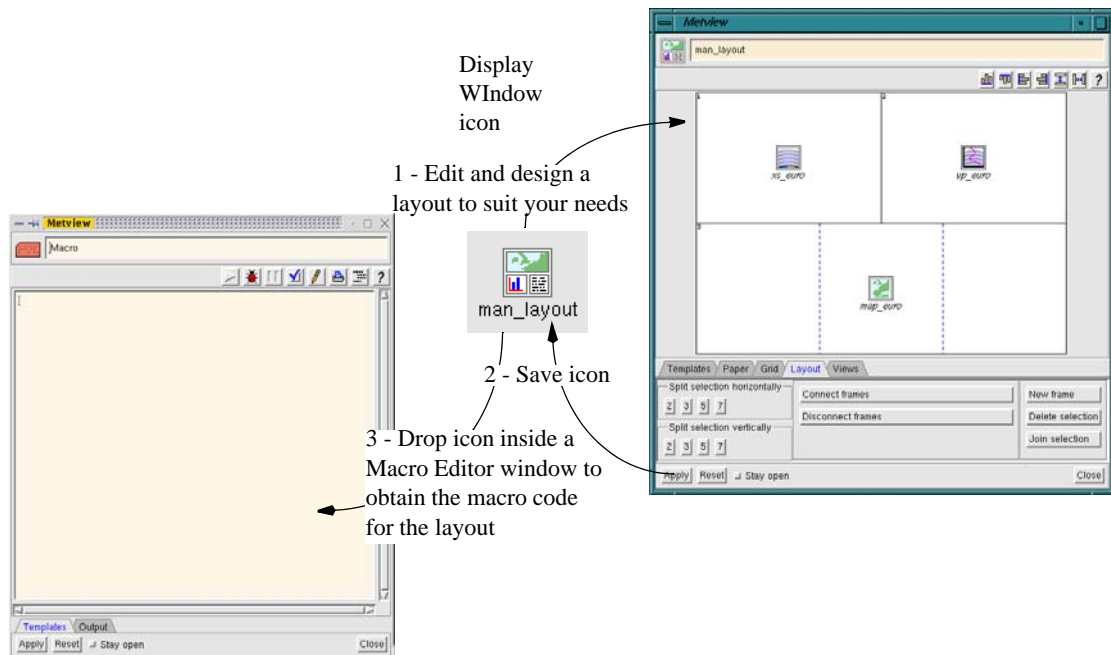


Figure II-7 : Using the Display Window icon to build a layout and generating the corresponding macro program. This macro should undergo some minor adjustments before usage

The steps (shown in Figure II-7) are :

- Edit a Display Window icon and prepare a layout to your liking, including the required views (see the chapter "The Display Layout" on page I- 71 for full details on layout design using this interactive icon editor).
- Save the icon once done
- Drop the Display Window icon inside a Macro editor window - this generates the macro code corresponding to the layout you have designed

From here you can keep on coding the remainder of the macro. If you intend to use the converted layout very frequently, you should make a function out of it or place it in a separate macro for inclusion in other macros (see "Building a Library of Functions" on page II- 47).

Assigning Data to Frames and Subframes

Display windows with multiple frames are used to plot several data units on the same view or the same data unit on different views. Hence you need to know how to assign a given data unit to a particular frame.

The frames composing a layout are specified in a list variable, input to the `pages` variable of the `plot_superpage()` function. A specific frame is addressed simply by indexing the display window definition. E.g. given the code :

```

page_left = plot_page(
    left :    0,
    right :   50
)
page_right = plot_page(
    left :   50,
    right :  100
)

dw = plot_superpage(
    pages :   [page_left,page_right]
)

plot(dw[1], data1)
plot(dw[2], data2)

```

The frame defined by `page_left` is addressed by `dw[1]`, while the frame defined defined by `page_right` is addressed by `dw[2]`, so `data1` is plotted on the left frame and `data2` on the right frame. In general :

```
display_window[n]
```

addresses the frame defined by the `n`th variable in the list of pages. So it all depends on the sequence in which you entered the elements in the list of pages and on how you address the display window - the function presented in "A macro function for layouts of MxN frames" on page II- 67 generates frames with numbers starting on the top-left frame, increasing rightwards and then downwards to conform with the convention adopted for the Display Window icon editor.

When plotting multiple data to a single frame layout which is divided in a number of subframes, *you cannot address individual subframes explicitly*. This does not mean you have no control over which subframe is taken up by which field - the subframes are specified in a list variable, input to the `subpages` variable of the `plot_page()` function. The rule is :

- When plotting a fieldset with N fields (or other multiple data unit) the first field goes to the subframe defined by the first element in the list of subframes. The second to the second element, etc.,

This means that by modifying the sequence of the elements in the list supplied to `subpages` in `plot_page()`, you can control which field goes where in the subframes.

```

one_page = plot_page(
    view      : map_euro,
    sub_pages : [top_left, top_right, bott_left, bott_right]
)

dw2 = plot_superpage(
    pages :   one_page
)

plot (dw2,Tgrib)

```

This plots the field `Tgrib[1]` in the topleft subframe, `Tgrib[2]` in the topright subframe, etc.,
But had you defined the subpage list with a different sequence :

```
one_page = plot_page(
    view      : map_euro,
    sub_pages : [bott_right, top_left, bott_left, top_right]
)
```

then `Tgrib[1]` would go to the bottom right subframe, `Tgrib[2]` to the topleft subframe, etc.,

Output Formats and Destinations

Overview

In Metview Macro the layout and plot instructions remain the same irrespective of the output media you want your visualisation to be produced on. By default, visualisations are produced in display windows on screen. For this output you needn't specify anything about the format and destination of your visualisation.

In addition to on-screen displays, Metview can produce visualisations in :

- PostScript
- PNG or JPEG

The choice of output format is up to the user, except when running Metview in batch mode. In this case, no on-screen output is possible and one of the above listed formats must be chosen instead.

If you decide to produce your visualisation in one of PS/PNG/JPEG formats, you must also choose one output destination. Metview allows the following output destinations :

- a preview program - your visualisation appears on screen under `ghostscript` for PS format or `xv` for JPEG and PNG
- a printer - your visualisation goes straight to a printer
- a file on disk - your visualisation is produced as a PS or JPEG or PNG file

The file destination is more flexible in that you can produce a preview or send it to a printer afterwards. For preview or printer destinations no user file is created. Note the differences in behaviour of the different file formats regarding subframes, as described in "Printing a visualisation" on page I- 112 and "Filenaming Conventions for JPEG and PNG Plots" on page I- 114.

Specifying output and destination

The specification of output format and destination is made through the complementary functions `output()` and `setoutput()` :

- `output()` - this function returns a definition variable which specifies the output format and destination

- `setoutput()` - this function sets as output format and destination to be used by a macro the ones specified by its single argument : a definition variable returned by `output()`

This is a convenient feature of Macro in that you can define several format/destination combinations through several calls to `output()` and then select one of them to be set depending on some conditions (e.g. if run in batch set a PS file, otherwise set on-screen). Note that the output destination specified with `setoutput()` will only be used if a display window is explicitly defined in the macro and included within the `plot()` function. **The `setoutput()` function must be called before any calls to the display window definition function, `plot_superpage()`, in order to take effect.**

The `output()` function can accept a wide and variable number of arguments. A few examples to cover the more frequent cases are presented next :

To output to screen - you need not do anything, but if you really want to make your code explicit you can use :

```
Screen = output (
           format      :   "screen"
           )
```

To output in PS format to a file :

```
ps_output = output(
  format      :   "postscript",
  destination :   "file",
  file_name   :   "my_output.ps"
  print_option : "all",
  printer     :   "ps_oa"
  )
```

To output in PS format to a preview :

```
ps_output = output(
  format      :   "postscript",
  destination :   "preview",
  preview_program : "ghostview"
  print_option : "visible",
  printer     :   "ps_oa"
  )
```

To output in PS format straight to a printer :

```
ps_output = output(
  format      :   "postscript",
  destination :   "printer",
  print_option : "all",
  ncopies     :   2,
  printer     :   "ps_oa"
  )
```

The variable `print_option` specifies whether all possible plots are plotted or only those which are visible - for instance if you are visualising a file with 10 fields on a 2x1 layout you can choose to print only the two which are visible at any one time or all the ten fields.

To create a PS file in a directory other than the one where the macro is running from, add the required path to the file name variable.

Note that you need to specify a printer even when producing a file using a preview, since it is the printer characteristics which determines whether the file will come out in black and white or colour.

The variable `ncopies` specifies the number of copies to print. This parameter has no effect when generating either a file or a preview; it works only when sending output directly to a printer.

To produce a JPEG file :

```
jpg_output = output(
    format           : "jpeg",
    destination      : "file",
    file_name        : "my_output.jpg",
    print_option     : "all",
    width_in_pixels  : 800,
    jpeg_quality_level : -1
)
```

To preview a JPEG format visualisation in XV :

```
jpg_output = output(
    format           : "jpeg",
    destination      : "preview",
    preview_program  : "xv",
    print_option     : "visible",
    width_in_pixels  : 800,
    jpeg_quality_level : -1
)
```

To output a JPEG format visualisation to a printer :

```
jpg_output = output(
    format           : "jpeg",
    destination      : "printer",
    preview_program  : "xv",
    print_option     : "visible",
    width_in_pixels  : 800,
    jpeg_quality_level : -1
    printer          : "ps_oa"
)
```

`output()` simply defines possible output formats and destinations. To specify the one to use you need the function `setoutput()`. Ideally, the choice is made depending on a macro implemented user choice, conditional on some parameter value.

The classic example is when users need the same macro program to be able to produce the same output in different formats/destinations depending on the run mode of the macro (see "Macro Run-Modes" on page II- 99 for details on macro run-modes) - typically if the macro is run in visualise mode, output to screen, if in batch mode output to a PS file :

```
# defines the available output formats (screen and ps)
screen = output (
    format      : "screen"
```

```
    )

    ps_file = output(
        format      : "postscript",
        destination : "file",
        file_name    : "my_output.ps"
        print_option : "all",
        printer      : "ps_oa"
    )

    # Checks the macro run mode
    mode = runmode()

    # sets output destination according to run-mode
    if mode = "batch" then
        setoutput(ps_file)
    else if mode = "visualise" then
        setoutput(screen)
    else
        fail("run mode not authorized - batch or visualise only")
    end if
```

Forcing a New Page

When plotting to a PostScript file, you can force a new page to be taken by calling the macro function `newpage()`. This can be useful for fine-tuning certain plots. For example, it is currently the case that a geographical curve can only be overlaid on the first of a stack of plots. Using the `newpage()` function, we can overcome this limitation with a little bit of code to manually loop through the fields:

```
dw = plot_superpage (...)
my_curve = curve (...)

for i= 1 to count(data_fields) do
    plot(dw[1], data_fields[i], my_curve)
    newpage(dw)
end for
```


USING FORTRAN IN MACRO

Overview

The ability to merge FORTRAN source and executables within macros is a very powerful feature of the Metview Macro language. It extends immensely the scope of the macro language and enables you to make efficient use of existing resources.

FORTRAN programs are used in tasks which cannot be achieved by means of a macro language function or combination of functions. This happens, for example, if the task requires such calculations which are a function of gridpoint positions. Alternatively, you may already have suitable FORTRAN code and the writing of the same task in Macro language would simply consume precious time.

Currently the FORTRAN-Metview Macro interface is supported for input data of the type `GRIB`, `number`, `string` and `vector`. Types `date`, `list` and `definition` are not yet supported as arguments or results of external FORTRAN functions. Types `BUFR`, `image` and `matrix` are awaiting implementation.

In this section we describe the requirements to write FORTRAN routines to use within Macro programs, the steps for their implementation (i.e. how to make them available to Macro programs) and provide practical examples.

Note that as of Metview 3.11.8, there exist two FORTRAN-Macro interfaces. The legacy interface which uses GRIBEX to handle GRIB fields is still available, and is described under "The Legacy FORTRAN Interface Routines" on page II- 83 for the purpose of maintaining legacy code. The new interface (Macro-Fortran Interface, or 'MFI' for short) uses GRIB_API to handle GRIB fields, and its function names are similar to those in the C/C++ interface (see "Using C/C++ in Macro" on page II- 91). This interface is preferred for handling GRIB data because it can cope with GRIB edition 2 as easily as GRIB edition 1 - see "The Macro-FORTRAN Interface (MFI) Routines" on page II- 79.

General Approach

Here we describe the general procedure required to write FORTRAN functions to be used within Metview Macro programs, whichever interface you use.

Most of your code will be composed of predefined routines. You only have to supply the routine which derives the required output. At its simplest, a FORTRAN program accepting GRIB input and producing GRIB output, and whose executable will be embedded in a Macro program, is composed of :

- a section where input is read and output prepared
- a section (loop) where the fields are loaded, expanded, validated and processing carried out (within the routine you supply)
- a section where the result is saved and set for output

The predefined routines include the *FORTRAN Interface Routines* which take care of input and output activities (in the broad sense of the word). If you use the legacy interface routines, then you will also use the GRIBEX() routine for the unpacking and packing of GRIB data. Otherwise, if you use the MFI routines, you will use various GRIB_API routines for handling the GRIB data.

Inlined or External

There are two ways of using a FORTRAN program from a macro: **inlined** and **external**. When a FORTRAN program is inlined, its source code is written directly into the macro's source file. This is the preferred way to use FORTRAN programs with Macro, as the user does not need to separately compile the program - this is done automatically when the macro is run, using compiler settings that are consistent with the Metview installation. The other option is to compile the FORTRAN program into an external executable and reference this from the macro. This has the disadvantage of being less portable (the executable will probably not be portable across platforms) and the user also has to ensure that the compiler settings are correct, and that the correct libraries are linked.

Interface routines

Your FORTRAN functions need to be able to accept input from the macro program and to produce output that can be used by the macro program in further calculations.

For this your FORTRAN function must use a suite of FORTRAN routines specially prepared for the purpose. They are called *FORTRAN Interface Routines*. A list of these routines and their function is presented in the following section. Among other things they :

- get the input arguments from the macro
- set the result type
- create, load and save fieldsets

If compiling your FORTRAN program into an external executable, you must link your FORTRAN function with the MARS and EMOS libraries in order to gain this functionality. By default, these libraries are placed in the \$METVIEW_DIR/lib directory. Additionally, if using the MFI routines, you should also link the GRIB_API libraries. The path to the GRIB_API include directory should also be supplied if it is not installed in a default location. Adding the following flag to the compile command should work in most cases :

```
-L$METVIEW_DIR/lib -lMars -lMvEmos -lgrib_api_f90 -lgrib_api
-I<grib_api_path>/include
```

For the legacy interface, only the MARS and EMOSLIB libraries are required:

```
-L$METVIEW_DIR/lib -lMars -lMvEmos
```

GRIB_API routines

If passing GRIB data using the MFI interface, then you will very probably need to use GRIB_API routines to handle the data. The example which follows shows an example of this, and more information can be found on the GRIB_API home page:

http://www.ecmwf.int/products/data/software/grib_api.html

GRIBEX() routine

If passing GRIB data using the legacy interface, then a further function you must use is the GRIBEX() routine which will

- decode GRIB headers
- expand (unpack) GRIB data (fieldsets)
- repack GRIB data (fieldsets).

Here only a synopsis of its arguments is provided. Detailed information, in particular the values that the arguments may take, can be found in the user guide of the MARS system, ECMWF Bulletin B6.7/2 "MARS USER GUIDE (For Data Retrieval)", section 4.1.1 - *Routines for packed format data*, pages 17 to 32; or in the Meteorological Bulletin M 1.9/3 "Encoding and decoding GRIB data (GRIBEX)". Note that the current sizes of the arrays should be checked against the current GRIBEX documentation at <http://www.ecmwf.int/publications/manuals/libraries/gribex/callGribex.html>.

```
CALL SUBROUTINE GRIBEX (KSEC0,KSEC1,KSEC2,PSEC2,KSEC3,
PSEC3,KSEC4,PSEC4,KLENP,KGRIB,KLENG,KWORD,HOPER,KRET)
```

```
K prefix - integer
P prefix - real
O prefix - logical
H prefix - character
```

```
KSEC0 - Size of GRIB message and GRIB edition number
KSEC1 - Integer parameters of Section 1 of GRIB code
KSEC2 - Integer parameters of Section 2 of GRIB code
PSEC2 - Real parameters for Section 2 of GRIB code
KSEC3 - Integer parameters of Section 3 of GRIB code
PSEC3 - Real parameters for Section 3 of GRIB code
KSEC4 - Integer parameters of Section 4 of GRIB code
PSEC4 - Array of data values (unpacked or to be packed)
KLENP - Length of array PSEC4
KGRIB - Array containing GRIB coded data
KLENG - Length of array KGRIB
KWORD - Number of words of KGRIB occupied by coded data
HOPER - Requested function : code/decode/return length/...
KRET - Response to error indicator
```

The Macro-FORTRAN Interface (MFI) Routines

The Macro-FORTRAN interface (MFI) routines enable your FORTRAN program to communicate with the macro program by handling all input and output operations.

- All the routines are prefixed with `mfi`. All the routines are suffixed by the type of the data they handle, for example `mfi_get_string`.
- Currently there is only the possibility to handle numbers as double precision reals; the legacy interface additionally included functions to handle them as single precision reals.

Getting arguments

`mfi_get_number(DOUBLE PRECISION)` - Get the next argument as a double precision real.

`mfi_get_string(CHARACTER(*))` - Get the next argument as a string

`mfi_get_fieldset(INTEGER, INTEGER)` - Get the next argument as a fieldset. The first argument returned is an id for the fieldset, used for calls to `mfi_load_one_grib`, `mfi_save_grib`, `mfi_return_fieldset` and `mfi_rewind_fieldset`. The second argument returned is the total number of fields in the fieldset.

`mfi_get_vector(REAL(*), INTEGER)` - Get the next argument as an array of reals. The first is the array to be filled (this must be declared to be large enough to hold the incoming data). The second argument acts as both input and output - initially, it should be set to the number of elements in the FORTRAN array. If the array is too small to hold all of the vector's elements, then as many as will fit are copied into the array. The second argument is set to the actual number of values copied into the array. **Note** that the ordering of these arguments is different from the legacy routine MGETV.

Setting results

These routines set the result type. If more than one result is set, the macro receives a list :

`mfi_return_number(DOUBLE PRECISION)` - Set next result as a double precision real.

`mfi_return_string(CHARACTER(*))` - Set next result as a string.

`mfi_return_fieldset(INTEGER)` - Set next result as a fieldset. The parameter is a fieldset id created by `mfi_new_fieldset` or `mfi_get_fieldset`.

`mfi_return_vector(REAL(*), INTEGER)` - Set next result as a vector. The first parameter is the array to be copied, the second the number of elements to set. **Note** that the ordering of these arguments is different from the legacy routine MSETV.

Handling fieldsets

These routines create, read and save fieldsets :

`mfi_new_fieldset(INTEGER)` - Create an empty fieldset. The parameter is an id to be used by `mfi_save_grib` and `mfi_return_fieldset`.

`mfi_load_one_grib(INTEGER, INTEGER)` - Read the next field of a fieldset. The first argument is the id returned by `mfi_get_fieldset`. The second argument is a GRIB id which can be used in subsequent calls to GRIB_API routines such as `grib_get`.

`mfi_save_grib(INTEGER, INTEGER)` - Save the next field of a fieldset. The first argument is the fieldset id returned by `mfi_new_fieldset`. The second argument is a GRIB_API GRIB id.

`mfi_rewind_fieldset(INTEGER)` - Rewind the fieldset. The argument is the id returned by `mfi_get_fieldset`.

Utility routines

`mfi_args(INTEGER, CHARACTER(*))` - Returns the number of arguments in the first argument and returns their type as a string in the second argument. A return value of "NNG" means two numbers and a fieldset.

`mfi_fail(CHARACTER(*))` - Abort and return a message to the macro.

Example of FORTRAN Function

Here we provide an example of FORTRAN externals, commented throughout so you can see which, how, and where each interface routine is used and where the user written routine fits. This FORTRAN program calculates the gradient of a scalar.

```

PROGRAM GRADIENT
USE grib_api
IMPLICIT NONE

INTEGER fieldset_in, fieldset_out, icnt
INTEGER grib_id, isize, istatus, i
INTEGER byte_size
REAL*8, ALLOCATABLE :: grib_in(:)
REAL*8, ALLOCATABLE :: grib_out_u(:)
REAL*8, ALLOCATABLE :: grib_out_v(:)

                                !-- GET FIRST ARGUMENT AS A FIELDSET.
                                !-- icnt IS THE NUMBER OF FIELDS
CALL mfi_get_fieldset( fieldset_in, icnt )

                                !-- CREATE A NEW OUTPUT FIELDSET
CALL mfi_new_fieldset( fieldset_out )

                                !-- LOOP ON FIELDS
DO i=1, icnt
                                !-- GET NEXT FIELD FROM INPUT FIELDSET
CALL mfi_load_one_grib( FIELDSET_IN, grib_id )

                                !-- ALLOCATE ARRAYS, GET FIELD VALUES
CALL grib_get_size( grib_id, 'values', isize )
ALLOCATE( grib_in(isize), grib_out_u(isize), grib_out_v(isize) )

CALL grib_get_real8_array( grib_id, 'values', grib_in, isize )

                                !-- VALIDATE AND DERIVE OUTPUT
CALL valid( grib_id )
CALL grad( grib_in, grib_out_u, grib_out_v )

                                !-- SET OUTPUT AS U COMPONENT OF WIND
CALL grib_set_real8_array( grib_id, 'values', grib_out_u, isize )
CALL grib_set_int( grib_id, 'paramId', 131 )

                                !-- ADD IT TO THE OUTPUT FIELDSET
CALL mfi_save_grib( fieldset_out, grib_id )

                                !-- SET OUTPUT AS V COMPONENT OF WIND

```

```

CALL grib_set_real8_array( grib_id, 'values', grib_out_v, isize )
CALL grib_set_int( grib_id, 'paramId', 132 )

                                !-- ADD IT TO THE OUTPUT FIELDSET
CALL mfi_save_grib( fieldset_out, grib_id )

                                !-- RELEASE MEMORY
CALL grib_release( grib_id )
DEALLOCATE( grib_in, grib_out_u, grib_out_v )

END DO

                                !-- RETURN THE RESULT
CALL mfi_return_fieldset( fieldset_out )

STOP
END

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!--
!-- USER ROUTINE TO CHECK VALIDITY OF INPUT FIELD
!-- VALID FOR A GLOBAL FIELD, LAT/LONG, 1.5 DEG GRID
!--

SUBROUTINE valid( grib_id )

USE grib_api
INTEGER grib_id
INTEGER ivalue
REAL*8  rvalue

CALL grib_get_int(grib_id, 'dataRepresentationType', ivalue)
IF( ivalue .NE. 0 ) CALL mfi_fail("GRID not lat/lon")

CALL grib_get_real8(grib_id, 'iDirectionIncrementInDegrees', rvalue)
IF( rvalue .NE. 1.5 ) CALL mfi_fail("GRID not 1.5/1.5")

CALL grib_get_real8(grib_id, 'jDirectionIncrementInDegrees', rvalue)
IF( rvalue .NE. 1.5 ) CALL mfi_fail("GRID not 1.5/1.5")

CALL grib_get_int( grib_id, 'Ni', ivalue )
IF( ivalue .NE. 240 ) CALL mfi_fail("GRID not global")

CALL grib_get_int( grib_id, 'Nj', ivalue )
IF( ivalue .NE. 121 ) CALL mfi_fail("GRID not global")

RETURN
END

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!--
!-- DERIVE GRADIENT OF INPUT FIELD F (BENITO ELVIRA, IM)
!-- FA = HORIZONTAL GRADIENT, FB = VERTICAL GRADIENT
!--

SUBROUTINE GRAD (F, FA, FB)

                                !-- DIMENSIONS CORRESPONDING TO 1.5 x 1.5 GRID
DIMENSION F(240,121), FA(240,121), FB(240,121)

```

```

PI = ACOS(-1.0)
RT = 6371000.0
CB = (RT*1.5*PI)/180.0

                                !-- COMPUTE HORIZONTAL GRADIENT
DO I = 1, 121

    C = COS( (90.0-I*1.5 + 1.5)*PI/180.0 )
    FA(1,i) = (F(2,i)-F(240,i)) / (2.0*C*CB)
    FA(240,i) = (F(1,i)-F(239,i)) / (2.0*C*CB)

    DO J = 2, 239
        FA(j,i) = (F(j+1,i)-F(j-1,i)) / (2.0*C*CB)
    END DO

END DO

                                !-- COMPUTE VERTICAL GRADIENT
DO I = 1, 240

    FB(i,1) = 0
    FB(i,121) = 0
    DO J = 2, 120
        FB(i,j) = (F(i,j+1)-F(i,j-1)) / (-2.0*CB)
    END DO

END DO

RETURN
END

```

Note that this function calculates the two components of the gradient of the input fieldset. These are saved separately and coded as wind components, so that each of these can be accessed separately in the macro for the calculation of the advection.

Examination of the above example shows that many of the FORTRAN functions you may need to write will be fairly similar as access to the input and creation of output is fairly standard. You have only to supply the processing routine (GRAD in our example).

The Legacy FORTRAN Interface Routines

The legacy FORTRAN interface routines enable your FORTRAN program to communicate with the macro program by handling all input and output operations. It is recommended that for new code you use the MFI routines (see "The Macro-FORTRAN Interface (MFI) Routines" on page II-79).

- All the routines are prefixed with M.
- All the routines are suffixed by the type of the data they handle:

```

G    for GRIB
N    for number(N2 for double precision)
B    for BUFR (not yet implemented)
I    for image (not yet implemented)
S    for string
V    for vector

```

M for matrix(not yet implemented)

Getting arguments

MGETN(REAL) - Get the next argument as a single precision real.

MGETN2(DOUBLE PRECISION) - Get the next argument as a double precision real.

MGETS(CHARACTER(*)) - Get the next argument as a string

MGETG(INTEGER, INTEGER) - Get the next argument as a fieldset. The first arguments is a pointer to the fieldset, used for calls to MSETG and MLOADG. The second argument is the total number of fields in the fieldset.

MGETV(INTEGER, REAL(*)) - Get the next argument as an array of reals. The first argument acts as both input and output - initially, it should be set to the number of elements in the FORTRAN array. The second is the array to be filled (this must be declared to be large enough to hold the incoming data). If the array is too small to hold all of the vector's elements, then as many as will fit are copied into the array. The first argument is set to the actual number of values copied into the array.

Setting results

These routines set the result type. If more than one result is set, the macro receives a list :

MSETN(REAL) - Set next result as a single precision real.

MSETN2(DOUBLE PRECISION) - Set next result as a double precision real.

MSETS(CHARACTER(*)) - Set next result as a string.

MSETG(INTEGER) - Set next result as a fieldset. The parameter is a pointer created by MNEWG or MGETG.

MSETV(INTEGER, REAL(*)) - Set next result as a vector. The first parameter is the number of elements to set, the second the array to be copied.

Handling fieldsets

These routines create, read and save fieldsets :

MNEWG(INTEGER) - Create an empty fieldset. The parameter is a pointer to be used with/by MSAVEG.

MLOADG(INTEGER, INTEGER(*), INTEGER) - Read the next field of a fieldset. The first argument is the pointer returned by MGETG. The second argument is a buffer big enough to hold a GRIB packed product. The third argument is the size of the product in words.

MSAVEG(INTEGER, INTEGER(*), INTEGER) - Save the next field of a fieldset. The first argument is the pointer returned by MNEWG. The second argument is a buffer containing a packed GRIB product. The third argument is the size of the product in words.

MREWG(INTEGER) - Rewind the fieldset. The argument is the pointer returned by MGETG. Note that this function is currently only available in FORTRAN source code that is inlined within a Metview macro - see "External Functions" on page II- 48 for details of inlined functions.

Utility routines

MARGS(INTEGER, CHARACTER(*)) - Returns the number of arguments in the first argument and returns their type as a string in the second argument. A return value of "NNG" means two numbers and a fieldset.

MFAIL(CHARACTER(*)) - Abort and return a message to the macro.

Example of FORTRAN Function

Here we provide an example of FORTRAN externals, commented throughout so you can see which, how, and where each interface routine is used and where the user written routine fits. This FORTRAN program calculates the gradient of a scalar.

```

PROGRAM GRADIENT
PARAMETER ( ISIZE=50000 )
DIMENSION IFIELD( ISIZE )
DIMENSION ISEC0( 2 )
DIMENSION ISEC1( 1024 )
DIMENSION ISEC2( 1024 )
DIMENSION ISEC3( 2 )
DIMENSION ISEC4( 512 )
DIMENSION ZSEC2( 512 )
DIMENSION ZSEC3( 2 )
DIMENSION ZSEC4( ISIZE )
DIMENSION ZSEC4A( ISIZE )
DIMENSION ZSEC4B( ISIZE )

C GET THE FIRST ARGUMENT AS A FIELDSET.
C ICNT IS THE NUMBER OF FIELDS
CALL MGETG( IGRIB1, ICNT )

C CREATE A NEW OUTPUT FIELDSET
CALL MNEWG( IGRIB2 )
IERR = 0

C LOOP ON FIELDS
DO 10 I=1, ICNT

C GET NEXT FIELD FROM INPUT FIELDSET
CALL MLOADG( IGRIB1, IFIELD, ISIZE )

C EXPAND IT
IPNTS = ISIZE
CALL GRIBEX ( ISEC0, ISEC1, ISEC2, ZSEC2, ISEC3,
+           ZSEC3, ISEC4, ZSEC4, IPNTS, IFIELD,
+           ISIZE, IWORD, 'D', IERR )

C VALIDATE AND DERIVE OUTPUT
CALL VALID( ISEC2 )

```

```

        CALL GRAD(ZSEC4,ZSEC4A,ZSEC4B)

C REPACK OUTPUT AS U COMPONENT OF WIND
      ISEC1(6) = 131
      CALL GRIBEX (ISEC0,ISEC1,ISEC2,ZSEC2,ISEC3,
+               ZSEC3,ISEC4,ZSEC4A,IPNTS,IFIELD,
+               ISIZE,IWORD,'C',IERR)

C ADD IT TO THE OUTPUT FIELDSET
      CALL MSAVEG(IGRIB2,IFIELD,IWORD)

C REPACK OUTPUT AS V COMPONENT OF WIND
      ISEC1(6) = 132
      CALL GRIBEX (ISEC0,ISEC1,ISEC2,ZSEC2,ISEC3,
+               ZSEC3,ISEC4,ZSEC4B,IPNTS,IFIELD,
+               ISIZE,IWORD,'C',IERR)

C ADD IT TO THE OUTPUT FIELDSET
      CALL MSAVEG(IGRIB2,IFIELD,IWORD)

10 CONTINUE

C SET RESULT
      CALL MSETG(IGRIB2)

      STOP
      END

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C USER ROUTINE TO CHECK VALIDITY OF INPUT FIELD
C VALID FOR A GLOBAL FIELD, LAT/LONG, 1.5 DEG GRID

      SUBROUTINE VALID(ISEC2)
      DIMENSION ISEC2(*)

      IF(ISEC2(1).NE.0) CALL MFAIL("GRID NOT LAT/LON")
      IF(ISEC2(9).NE.1500)CALL MFAIL("GRID NOT 1.5/1.5")
      IF(ISEC2(10).NE.1500)CALL MFAIL("GRID NOT 1.5/1.5")
      IF(ISEC2(2).NE.240) CALL MFAIL("GRID NOT GLOBAL")
      IF(ISEC2(3).NE.121) CALL MFAIL("GRID NOT GLOBAL")

      RETURN
      END

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C DERIVE GRADIENT OF INPUT FIELD F (BENITO ELVIRA, IM)
C FA = HORIZONTAL GRADIENT, FB = VERTICAL GRADIENT

      SUBROUTINE GRAD (F, FA, FB)

C DIMENSIONS CORRESPONDING TO 1.5 X 1.5 GRID
      DIMENSION F(240,121), FA(240,121), FB(240,121)

      PI = ACOS(-1.0)
      RT = 6371000.0
      CB = (RT*1.5*PI)/180.0

C COMPUTE HORIZONTAL GRADIENT

```

```

DO 10 I = 1, 121

    C = COS( (90.0-I*1.5 + 1.5)*PI/180.0 )
    FA(1,i) = (F(2,i)-F(240,i)) / (2.0*C*CB)
    FA(240,i) = (F(1,i)-F(239,i)) / (2.0*C*CB)
DO 20 J = 2, 239
    FA(j,i) = (F(j+1,i)-F(j-1,i)) / (2.0*C*CB)

10 CONTINUE

C COMPUTE VERTICAL GRADIENT
DO 30 I = 1, 240

    FB(i,1) = 0
    FB(i,121) = 0
DO 40 J = 2, 120
    FB(i,j) = (F(i,j+1)-F(i,j-1)) / (-2.0*CB)

30 CONTINUE

RETURN
END

```

Note that this function calculates the two components of the gradient of the input fieldset. These are saved separately and coded as wind components, so that each of these can be accessed separately in the macro for the calculation of the advection.

Examination of the above example shows that many of the FORTRAN functions you may need to write will be fairly similar as access to the input and creation of output is fairly standard. You have only to supply the processing routine (GRAD in our example).

Implementing FORTRAN Functions

Once you have written your FORTRAN routines, you have to either embed them into your macro code, or debug/compile/link them to obtain an executable.

To embed FORTRAN functions directly into macro code, surround the FORTRAN code with the following two lines (assuming here that the program is to be referred to as 'gradientb' and takes one fieldset at its argument) :

```

extern gradientb(f:fieldset) "fortran90" inline
... (FORTRAN code here)
end inline

```

This inlined code may now be put directly into your macro program that uses it, or else in a macro file by itself so that it will be available for use in other macros. See "External Functions" on page II-48 for more details of inlining FORTRAN functions.

To make FORTRAN functions that exist in externally-compiled executables available for use in macro programs, you can use the `extern` keyword in your macro program as shown in the following code that refers to the gradient example we are looking at:

```

extern gradientb(f:fieldset) "/home/xy/xyz/metview/progs/gradient"

```

Alternatively, it is possible to load the FORTRAN routine, whether inlined or executable, automatically by :

- placing the FORTRAN executable or inlined code in the `/home/xy/xyz/metview/Metview/Macro` folder, so that it is loaded automatically, and hence used without the need for an explicit declaration (see "Building a Library of Functions" on page II- 47). Only you can use these functions

or by

- having the FORTRAN executable or inlined code placed in the system wide Macro folder by whoever has write access to it (your institution's Metview developer/administrator). Then, every user will have access to this function.

Once this is done, the FORTRAN function is used exactly like any other Macro function, with no difference in the syntax. In effect, if the FORTRAN function is loaded automatically and hence there are no telltale declarations in the body of the macro program, it is not possible to say if it is a FORTRAN or a Macro function.

Note

In order for Metview Macro to determine whether a file in a Macro folder is a FORTRAN executable or a macro, it checks the return value of the UNIX `file` command on the file. If it includes the word 'executable', then it is assumed to be a FORTRAN program; otherwise, it is assumed to be a Metview macro (which can contain inlined FORTRAN code).

Implementation Example

The implementation of a FORTRAN routine is achieved by declaring the function (or loading automatically) and using it as you would any Macro function. To declare the FORTRAN executable `gradientb` as a Macro function called `gradient`:

```
extern gradient(f:fieldset) ".../metview/.../gradientb"
```

However if you place the executable program in the Macro system folder or if it resides in the global system-wide Macro folder, it will be detected automatically and you do not need to use the line of code above. An example of implementation of the FORTRAN function `gradientb` is presented next :

```
# This macro calculates the advection of specific humidity
# Retrieve the specific humidity
q = retrieve(
    date :    -1,
    param:    "q",
    level:    700,
    grid :    [1.5,1.5]
)

# Get the u component of the wind
u = retrieve(
    date :    -1,
    param:    "u",
```

```

        level:      700,
        grid :      [1.5,1.5]
    )

# Get the v component of the wind
v = retrieve(
    date :      -1,
    param:      "v",
    level:      700,
    grid :      [1.5,1.5]
)

# Compute the gradient of Q
q = gradient(q)

# Compute the advection of Q
a = q[1]*u + q[2]*v
a = -a * (10 ^ 8) # units will be 10e-8 (kg/kg)/sec

# Plot positive advection in blue, negative in red
contour_common = (
    contour_level_selection_type : "interval",
    contour_interval             :      3,
    contour_label                :      "on",
    contour_label_frequency      :      1,
    contour_label_quality        :      "high",
    contour_label_height         :      0.25,
    contour_hilo                 :      "off"
)

cont_n = pcont(
    contour_common,
    contour_max_level           :      -0.0001,
    contour_line_colour         :      "red",
    contour_label_colour        :      "red",
    contour_lo_colour           :      "red"
)

cont_p = pcont(
    contour_common,
    contour_min_level           :      0.0001,
    contour_line_colour         :      "blue",
    contour_label_colour        :      "blue",
    contour_hi_colour           :      "blue"
)

plot(a,cont_p,cont_n)

```

Note that you have just used the `gradient` function (implemented as a FORTRAN executable, `gradientb`) as if it was a Macro function - indeed unless the user is told (or has done it herself) there is no way to know whether `gradient` is a genuine Macro function or a FORTRAN executable.

Debugging the FORTRAN Functions

It is possible to run your embedded FORTRAN function under a debugger when it is executed. The simplest way is to run metview with the command-line option

```
-mfdbg=<myFavouriteDebugger>
```

where <myFavouriteDebugger> is the command for running your chosen debugger. This flag internally sets the environment variables MACRO_FORTRAN_DBG, MACRO_EXTRA_F90_FLAGS and MACRO_EXTRA_F77_FLAGS. If you wish to do this through your own environment instead of using the command-line flag, then the following shows a sample command-line sequence for manually setting up the environment and then starting metview:

```
setenv MACRO_FORTRAN_DBG tv6
setenv MACRO_EXTRA_F90_FLAGS -g
metview
```

In the case of inlined Fortran code, the source code is visible in the debugger by default; for external Fortran executables, you will need to set the source directory in the debugger.

USING C/C++ IN MACRO

Overview

Metview is capable of incorporating C/C++ source and executables within macros, similar to the way in which it incorporates FORTRAN. Much of the introductory text describing the FORTRAN interface is also true of the C interface and will not be repeated here (see "Using FORTRAN in Macro" on page II- 77). The Interface Routines however are different, and the intention is to use GRIB_API instead of GRIBEX to access GRIB data. The Interface Routines return a `grib_handle`, which can be used with most GRIB_API functions. You may wish to visit http://www.ecmwf.int/products/data/software/grib_api.html for more information on GRIB_API. Example 2 (see "Example 2 - C" on page II- 95) shows how to obtain an array of values from a field.

Currently the C-Metview Macro interface supports data of the type `fieldset`, `number`, `string` and `vector`. Types `date`, `list` and `definition` are not yet supported as arguments or results of external C functions. Types `BUFR`, `image` and `matrix` are awaiting implementation.

Writing Macro functions in C/C++

This is a list of C/C++ callable interface functions needed to write Macro functions in C/C++. All interface function names start with "mci_" (short for "Macro C Interface").

Programs using these interface functions need to include the header file 'macro_api.h'. For inline C/C++ programs the path to this header file does not need to be given.

Note that for some input arguments these interfaces allocate memory which is left to the calling program to free - this is mentioned in the function descriptions. Usually the memory allocated is not very large and in most cases can be left to be freed automatically when the function exits.

Numbers

The Macro number type is a double floating point. Also integer values are stored in 'double' variables and thus the number interface is only for this type.

```
double mci_get_number();
```

Reads the next argument (must be a number). Calls 'abort()' if the argument is not a number.

```
void mci_return_number( double d );
```

Returns double value 'd' to the calling macro.

Text Strings

The Macro string interface is for null terminated "C-style strings".

```
const char* mci_get_string();
```

Reads the next argument (must be a string). Note that for each string argument, `mci_get_string` reuses or reallocates the internally allocated memory. Thus, if there are several string arguments, the C/C++ program should make a copy of the string. For instance, in C by calling `mci_get_string` as an argument to function `strdup`:

```
char* s = strdup(mci_get_string());
```

and in C++ using STL string

```
std::string s(mci_get_string());
```

Calls 'abort()' if the argument is not a string.

```
void mci_return_string( const char* s );
```

Returns null terminated string 's' to the calling macro.

Vectors

Macro vectors are one dimensional double floating point arrays.

```
void mci_get_vector( double** vec, int* length );
```

Reads the next argument (must be a vector) into 'vec' and the length of the vector into 'length'. The function allocates memory for 'vec' and it is up to the calling program to free this memory.

Calls 'abort()' if the argument is not a vector.

```
void mci_return_vector( double* vec, int length );
```

Returns vector 'vec', of length 'length', to the calling macro.

Input Fieldsets

Note: as of Metview version 3.11.8, the function 'mci_get_grib_id_ptr' has been replaced by 'mci_get_fieldset', 'mci_rewind_grib' by 'mci_rewind_fieldset', 'mci_new_grib_id_ptr' by 'mci_new_fieldset' and 'mci_return_grib' by 'mci_return_fieldset'. The former function names will still work, but a warning message will be issued. There is no functional difference, but the new names are clearer.

Input fieldsets are accessed by first calling 'mci_get_fieldset' to get a "fieldset handle" and then, using the handle, to call 'mci_load_one_grib' to get a "grib_api handle" to access the fields, one by one. If the input fieldset needs to be read twice, it can be rewound by calling 'mci_rewind_fieldset'.


```
void* mci_get_fieldset( int* field_count );
```

Reads the next argument (must be a fieldset) and returns a void pointer to an internal data structure. Function allocates memory for the data structure and it is up to the calling program to free this memory. Calls 'abort()' if the argument is not a fieldset.

```
grib_handle* mci_load_one_grib( void* fieldset_ptr );
```

Returns a "grib_api handle" to the next field in a fieldset pointed to by the void pointer returned by a call to 'mci_get_grib_id_ptr'. The function allocates memory for the grib_api handle and it is up to the calling program to free this memory. Returns a NULL pointer if there are no more fields in the fieldset.

```
void mci_rewind_fieldset( void* fieldset_ptr );
```

Rewinds the fieldset pointed to by the void pointer returned by a call to 'mci_get_fieldset'.

Output Fieldsets

Output fieldsets are handled by first calling 'mci_new_fieldset' to create a "fieldset handle" for an empty fieldset and then, using the handle, to call 'mci_save_grib' to add new fields (handled by 'grib_api' handles) into the fieldset, one by one. Once the output fieldset is ready, a call to function 'mci_return_grib' will prepare the output fieldset for returning it to the calling macro.

```
void* mci_new_fieldset();
```

Creates an internal data structure for an output fieldset and returns a void pointer to the structure. It is up to the calling program to free the void pointer.

```
void mci_save_grib( void* fieldset_ptr, grib_handle* gh );
```

Adds a new field into the output fieldset pointed to by 'fieldset_ptr' (which was created by calling 'mci_new_fieldset').

```
void mci_return_fieldset( void* fieldset_ptr );
```

Returns the output fieldset to the calling macro.

Miscellaneous

```
int mci_arg_count();
```

Returns the number of input arguments in the function call.

```
const char* mci_args();
```

Returns a null terminated text string that contains one character per each argument in the function call. Each character represents the type of the corresponding input argument: 'N' for a number, 'S' for a string, 'G' for a fieldset (GRIB), and 'V' for a vector.

```
void mci_fail( const char* msg );
```

Forces the function (and the calling macro) to fail with message 'msg'.

Macro Examples Using the C Interface

Example 1 - C++

This is an example of a macro function that takes three arguments, changes them a little, and returns the modified values. There are no comments as the code is supposed to be self-explanatory.

```
# Metview Macro

extern ctest(n:number, s:string, v:vector) "C++" inline

#include <iostream>
#include "macro_api.h"

int main()
{
    int n = mci_arg_count();
    const char* ss = mci_args();
    std::cout << "Arg count: " << n
              << ", arg types: " << ss
              << std::endl;

    double d = mci_get_number();
    std::cout << "Number:      " << d << std::endl;

    std::string s = mci_get_string();
    std::cout << "String:      '" << s << "'" << std::endl;

    double* vec;
    int len;
    mci_get_vector( &vec, &len );
    std::cout << "Vector length: " << len
              << ", last elem: " << vec[len-1]
```

```

        << std::endl;

        mci_return_number( 2*d );

        s[0] = '#';
        mci_return_string(s.c_str());

        for( int i=0; i<len-2; ++i )
            vec[i] = 2*vec[i];

        mci_return_vector( vec, len-2 );
    }
end inline

# -----

n = 3.21
s = "Hello"
vec1 = | 1, 2, 3, 4, 5 |

r = ctest(n,s,vec1)
print( r )

vec2 = | 101, 102, 103, 104, 105, 106, 7777 |

print( ctest(2.345,"world",vec2) )

```

The output from this macro is:

```

Arg count: 3, arg types: NSV
Number:    3.21
String:    'Hello'
Vector length: 5, last elem: 5
[6.42,"#ello",|2,4,6|]
Arg count: 3, arg types: NSV
Number:    2.345
String:    'world'
Vector length: 7, last elem: 7777
[4.69,"#orld",|202,204,206,208,210|]

```

Example 2 - C

This is a dummy example as the inline C code function does not return any values. The function simply tries to extract grid point values from the first field (using 'grib_api' functions), and then prints whether it succeeded in this or not. It returns the number of values present in the first field.

```

#Metview Macro

extern grib_test(f:fieldset) "C" inline
#include <stdio.h>
#include <string.h>
#include "macro_api.h"

int main()
{
    grib_handle* gh = NULL;

```

```

void*   fieldset = NULL;

int   fcnt   = 0;   /*-- field count --*/
int   ret    = 0;   /*-- function return value --*/
size_t len   = 0;   /*-- number of grid point values --*/
double* vals = NULL; /*-- array for grid point values --*/

fieldset = mci_get_fieldset(&fcnt);

gh = mci_load_one_grib( fieldset );

grib_get_size(gh,"values",&len);
vals = (double*)malloc(len*sizeof(double));
printf("GRID size: %d\n",len);

ret = grib_get_double_array(gh,"values",vals,&len);

if( ret == GRIB_SUCCESS )
    printf("-- got the values !!! --\n" );
else
    printf(">>> grib_get_double_array returned %d\n", ret );

free(vals);
grib_handle_delete(gh);
free(grib_id);

mci_return_number(len);
}
end inline

# -----

fs = retrieve(levelist:[1000,850,500])
r = grib_test(fs)
print (r)

```

Example 3 - C++

This example is an extension of the C language example. This example reads all fields from the input fieldset, checks that it can extract the grid point values, and simply returns every second field from the input fieldset.

```

# Metview Macro

extern grib_test(f:fieldset) "C++" inline
#include <iostream>
#include <string.h>
#include "macro_api.h"

int main()
{
    grib_handle* gh   = NULL;   /*-- GRIB handle (grib_api)
    void* fieldset_in = NULL;   /*-- input fieldset handle

```

```

void*   fieldset_out = NULL; /-- return fieldset handle

int    fcnt    = 0;          /-- field count
int    ret     = 0;          /-- function return value
size_t len    = 0;          /-- number of grid point values
double* vals  = NULL;       /-- array for grid point values

fieldset_in = mci_get_fieldset(&fcnt); /-- get input fieldset ptr
std::cout << "Got fieldset ptr, file contains " << fcnt << " fields"
<< std::endl;

fieldset_out = mci_new_fieldset(); /-- create return fieldset ptr

for( int i=0; i<fcnt; ++i )
{
    gh = mci_load_one_grib( fieldset_in ); /-- get one input field into
// GRIB handle

    grib_get_size(gh,"values",&len); /-- enquire field size
    std::cout << "GRID size: " << len;
    vals = new double[len]; /-- get memory for values

    ret = grib_get_double_array(gh,"values",vals,&len); /-- get
values

    if( ret == GRIB_SUCCESS ) /-- OK?
        std::cout << ", got the values!!!";
    else
    {
        std::cout << ", but grib_get_double_array() returned " << ret <<
std::endl;
        mci_fail( "Unable to get values => FAIL" );
    }

    if( i % 2 ) /-- return every second field
    {
        mci_save_grib( fieldset_out, gh ); /-- add field to be returned
        std::cout << " <--- returned field (mci_save_grib)";
    }
    std::cout << std::endl;

    delete [] vals; /-- free values array
    grib_handle_delete(gh); /-- free current GRIB handle
}

mci_return_fieldset( fieldset_out ); /-- return selected fields

delete fieldset_in; /-- free input GRIB id ptr
delete fieldset_out; /-- free return GRIB id ptr
}
end inline

# -----

fs = read("/home/graphics/cgx/Examples_for_New_Users/user guide data")
print(fs)

r = grib_test(fs)
print("Function returned ",r)

```

```
plot(r)
```

Inlined or External

There are two ways of using a C/C++ program from a macro: **inlined** and **external**. When a program is inlined, its source code is written directly into the macro's source file. This is the preferred way to use C/C++ programs with Macro, as the user does not need to separately compile the program - this is done automatically when the macro is run, using compiler settings that are consistent with the Metview installation. The other option is to compile the C/C++ program into an external executable and reference this from the macro. This has the disadvantage of being less portable (the executable will probably not be portable across platforms) and the user also has to ensure that the compiler settings are correct, and that the correct libraries are linked.

If compiling your C/C++ program into an external executable, you must link with the MARS, EMOS and GRIB_API libraries in order to gain this functionality. By default, these libraries are placed in the \$METVIEW_DIR/lib directory. Adding the following flags to the compile command should work in most cases :

```
-I $METVIEW_DIR/lib -L $METVIEW_DIR/lib -l Mars -l MvEmos -l grib_api
```

You may also have to add '-l jasper' if you have enabled jpeg2000 encoding in your GRIB_API library.

To give an example, if we were to externally compile Example 3, then the Macro would now look like this:

```
# Metview Macro

extern grib_test(f:fieldset) "C++"

fs = read("/home/graphics/cgx/Examples_for_New_Users/user guide data")
print(fs)

r = grib_test(fs)
print("Function returned ",r)
plot(r)
```

If the executable is not a file called grib_test in the current directory, then you can specify a path to it:

```
extern grib_test(f:fieldset) "C++" '/x/y/z/grib_test'
```

INTERFACE TO THE METVIEW SYSTEM

Metview Macro has functions which provide an interface to the Metview system. A macro can determine its own run mode (i.e. how the running of the macro was triggered), it can access the data cache for reading and writing, and can control its exit upon some error or condition. A synopsis of the functions involved in these processes is presented in "Macro System Functions" on page II-203.

Macro Run-Modes

Overview

A Macro program can be run in several ways :

- by choosing one of the following options from its icon menu :
 - Execute
 - Visualise
 - Save
 - Examine
- by dropping the icon in a display window
- by passing it to a Metview session run in batch

Each of the above ways of running a macro corresponds to a so called **run-mode**. Which of the different macro run modes you choose may have an implication on the outcome of the macro. The available macro run modes are :

- Execute** - through the Execute icon menu option
- Visualise** - through the Visualise icon menu option
- Save** - through the Save icon menu option
- Examine** - through the Examine icon menu option
- Prepare** - dropping an icon inside a display window
- Edit** - running a macro through a MacroParameters user interface
- Batch** - running the macro in a Metview batch session

If you code a particular action explicitly in the macro - e.g. use the `plot()` function to plot some data, or the `write()` or `append()` functions to save a fieldset to a GRIB file - it does not matter how you run the macro (which run mode the macro is under), the outcome of the macro is as you specified in the program code.

So when does the run-mode have a bearing on the macro outcome? The two situations in which this can arise are :

- when the macro program only action is to return something to Metview
- when the macro program checks its own run mode and codes different outcomes for each run mode

These two situations are described next.

Macro returns to the Metview interface

One way to make the macro outcome depend on the run mode is by means of the `return` action. To use it, you do not specify any output action - e.g. no calls to `plot()` or `write()` or `append()` - you simply return something (this something being mostly some data), as in this simple example :

```
r = retrieve(param : "z", level: 500)
return r
```

In this example, when you run the macro, it returns the data (a fieldset) to the Metview User Interface which sends it to the Metview module that can handle the action specified by the run-mode :

For run mode **Execute**, the macro return is actually ignored. Any result which the macro is to achieve must be explicitly coded within the macro. The code above simply retrieves the data and places it in a cache.

For run modes **Drop** and **Visualise**, the result is sent to the Visualisation module. The Visualisation module expects either data (as above), or a list of data and visual definition(s) such as :

```
r1 = retrieve(param: "z", level: 500)
p1 = pcont(contour_line_colour: "red")
return [r1,p1]
```

For run mode **Save**, the returned data is saved; the first example code would create a Z500 field as a GRIB file on disk.

For run mode **Examine**, the returned data is examined; the first example code would launch an Examine window for this field. Note that you can only use Examine for GRIB data returns.

What you return must be consistent with the run-mode you choose. For instance, if your macro returns a visual definition only, the visualise, save and examine run modes will not produce any output - visual definitions can't be saved or examined, nor visualised on their own. However you could drop the macro in a display window - the returned visual definition would be applied to the data residing in there (assuming suitability).

Run-mode dependent outcomes

A macro can find its own run-mode (the action with which it is run) by a call to the `runmode()` function or by using handlers. With the `runmode()` function you can :

- derive the name of the action used to run the macro - call the `runmode()` function with no arguments :

```
mode = runmode()
```

in this case the function returns a string, the name of the run-mode the macro was run under (mode is `execute` if the macro is run with **Execute**, ...)

- check the action used to run the macro against a pre-specified run mode - call the `runmode()` function with a single string argument (the run-mode to be verified)


```
is_visualise = runmode("visualise")

if is_visualise then
    # Code in case macro is visualised
    (...)
else
    # Code in case macro is not visualised
    (...)
end if
```

`is_visualise` is set to 1 if the action is *Visualise* and 0 otherwise

Hence, with a correct use of `runmode()` you can prevent any unwanted outcome from a wrong choice of action and/or make the same macro provide different outcomes - e.g. produce a visualisation in a PS file if called with **Execute**, on-screen if called with **Visualise**, saving a derived fieldset if called with **Save**, etc.,. An example was presented in "Specifying output and destination" on page II-73.

Alternatively to the `runmode()` function, you can use special functions called "handlers" to isolate parts of the code one from the other. The code above could just as easily have been written using handlers :

```
on execute
    # code to "visualise" to ps file
    (...)
end execute

on visualise
    plot(x)
end visualise

etc...
```

Remember that handlers are functions, hence variables defined outside the handlers should be defined as global variables, otherwise you have to redefine them inside each handler.

Using a Data Cache in Macro

The Metview system puts its results in a cache. When an icon is executed, its output is stored in the cache, using the name of the icon. If the icon is executed later, the data is fetched from the cache. If the icon is modified, by editing its content, the associated data is removed from the cache.

The only Metview application that does not follow this scheme is the macro player, because the outcome of the macro may have to differ between two consecutive runs (the macro may use the current time to produce its results).

Nevertheless, you can explicitly use the cache by using the functions `store()` and `fetch()`.

As the name implies, `store()` puts data of your choice in the cache. Inputs to `store()` are the name you will cache your data under and the variable containing whatever it is you are caching (fieldsets, geopoints, ...).

```

# Retrieve some data
u = retrieve(date: -1, param: "u", grid :[2,2])
v = retrieve(date: -1, param: "v", grid :[2,2])

# Compute something
s = sqrt(u*u+v*v)

# Store the result
store("wind speed",s)

```

The function `fetch()` gets data from the cache and assigns it to some variable. Input to `fetch()` is simply the name under which you cached your data. The following line of code (which would be in another macro program) retrieves the data cached in the previous example and stores it in a variable which can then be plotted or operated upon.

```

# Retrieve previously cached result, store in s
s = fetch("wind speed")

```

One problem with this approach is that you can never be 100% sure that the data is in the cache as the Metview cache can be cleared at any time, when disk space needs to be recovered. So your code should not assume the existence of any data in the cache and hence should be able to recreate any result, should the cache have been cleared. It is straightforward to implement this, given that the `fetch()` function returns `nil` if the data is not in the cache :

```

# Retrieve data from the cache
s = fetch("wind speed")

# If it was not there, compute it
if s = nil then
  u = retrieve(date: -1, param: "u", grid :[2,2])
  v = retrieve(date: -1, param: "v", grid :[2,2])
  s = sqrt(u*u+v*v)
end if

# Code using s
(...)

```

Because Metview uses this scheme to cache the results of the icons in the user interface, you must be careful in choosing the name you give to cached data. Metview names its icons in a similar way to the UNIX file system. An icon named Retrieval in the main workspace will be cached as `/Retrieval`. An icon Retrieval in a folder Work will be named `/Work/Retrieval`. So, if you avoid names starting with a `/`, there may be no clashes.

A useful addition to the above couple of functions is the function `name()`. This function returns the name of the macro being executed. Below, is an example of its usage in conjunction with `store()` and `fetch()` :

```

# Check if this macro has already been executed
s = fetch(name())
if s <> nil then
  return s
end if

```

```
# Otherwise, compute s
u = retrieve(date : -1, param : "u", grid : [2,2])
v = retrieve(date : -1, param : "v", grid : [2,2])
s = sqrt(u*u+v*v)

# Store the result under the name of the macro
store(name(),s)

# ...and return it
return s
```

If you use this scheme, executing the same macro a second time will simply return the result cached during the first run. Editing the macro will automatically remove the data from the cache. So if the changes lead to different results, the macro will not return outdated results.

Exiting a Macro Program

Macro provides two functions to halt the execution of a macro program - `stop()` and `fail()`. Their only argument is a string which should contain a suitable (error) message. They operate in different ways :

- `stop()` simply exits the macro program and signals successful completion (i.e. macro icon name turns green)
- `fail()` forces a failure of the macro program and the macro icon name turns red. It also outputs an extra message (Line n : Macro failed), where n is the line number.

Hence, use `fail()` to exit Macro on error situations :

```
if (not(exist(my_grib_file.grb))) then
    fail("specified input file does not exist!")
end if
```

and `stop()` to halt execution upon some condition being met :

```
if (n_iter > max_n_iter) then
    stop("maximum number of iterations reached")
end if
```

INTERFACE TO THE UNIX SYSTEM

The Macro language provides you with a set of functions to interface to the UNIX operating system. These provide access to `shell` commands, change priority of the macro execution, check the contents of the cache, retrieve and set the value of environment variables, etc.. For a short listing see "UNIX Interfacing Functions" on page II- 201.

Controlling Macro Program Runs

You have some degree of control over a macro program execution - you may stop it mid-way through its execution for a given number of seconds using the `sleep()` function.

```
# Stops for 30 seconds
sleep(30)
```

You may also lower the priority of the macro relative to other processes with the `nice()` system call. You can obtain more detailed information from the `nice()` UNIX man page (type `man nice` at the UNIX prompt).

Access to the Shell

This function takes any UNIX command as parameter. The numbers and dates are first converted to string (for more information about conversions from date to string see "Converting date components" on page II- 13).

`shell()` returns a number, the exit status of the command invoked.

Access to Environment Variables

Metview allows you to check and set the value of environment variables, using the functions `getenv()` and `putenv()`.

```
# if PLOT_DATE not set, set to yesterday
pdate = getenv("PLOT_DATE")
if pdate = "" then
    yesterday = " " & yyyyymmdd(date(-1))
    putenv("PLOT_DATE", yesterday)
end if
```

BUILDING A MACRO USER INTERFACE

This section covers ways in which you can build a user interface. User interfaces are useful to provide generality to Macro programs. This means that the same Macro program used for a given task will be able to accept a variety of input parameters which will be provided via an icon editor window like interface.

The user interface is coded by the user using *input element functions* which specify which input elements to use, their input parameter names, default values, etc.

To obtain the actual user interface you either :

- use a function called `dialog()` that takes what the input element functions return to build a user interface ("open a dialogue"); this user interface is volatile in that it is rebuilt every time the macro is executed and input must be specified anew everytime

or

- use a Metview icon called Macro with Parameters which accepts as input a Macro program containing the input element functions and draws a permanent user interface, which remains associated with the icon and where the values you last specified remain in place

List of Input Element Functions

Input element functions return definitions that specify editor window input elements, such as text fields, sliders, fields, etc.,. There is one such function for each type of input element. In general you can define the name of the input parameter they are to be associated with, limit the range of values for input, provide help buttons, default values, etc.,.

The input to the input element functions is provided as a list of named variables, which is to say they accept definitions as input.

Input element functions include :

`any()` - for any alphanumeric input
`colour()` - for a colour selection tool
`icon()` - for an icon drop area
`slider()` - for a slider
`option_menu()` - for a multiple choice menu
`toggle()` - for an On/Off control

any(...)

Creates a text field input element for general alphanumeric input.

name - The name of parameter. Default is "any".

values - List of possible values. Default is anything.

help - Provides a help button next to the text field. Specify `help_input` to obtain a coordinate assist button and `help_script` to specify a script whose output will be displayed on-screen.

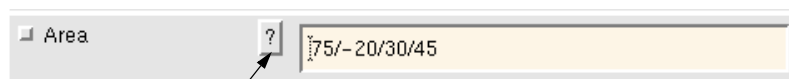
input_type - Specifies the type of coordinate assist button to provide - area, line or point. This is only applicable for help set to `help_input`.

help_script_command - Specifies a script to be run; its output will be displayed to the user. This can be an inline script stored as a string or the path to an external script file. Inline commands may be declared as simple strings or as multiline strings using the `inline` keyword - see "Strings" on page II- 9. This is only applicable for help set to `help_script`.

default - The initial value. Default is "default".

The following example code defines a text field meant to specify a list of Lat/Long values, separated by forward slashes (area definition) with an area geography help button.

```
a = any(
    name      : "area",
    exclusive : "false",
    values    : ["*", "/"],
    help      : "help_input",
    input_type : "area",
    default   : [75,-20,30,45]
)
```



Click-left for interactive area selection

The following code shows two uses of a help script. The first one displays help relevant to a particular user interface element and uses a simple inline script. The second is a more general help button which calls a script which will output instructions on using the macro.

```
ui_date = any
(
    name      : "Date",
    help      : "help_script",
    help_script_command : 'echo Dates must be in YYYYMMDD format',
    default   : 20040101
)

ui_help = any
(
    name      : "help",
    default   : "click here for help",
    help      : "help_script",
    help_script_command : "/home/.../macro_help.sh"
)
```

colour(...)

Creates a colour selection widget. This is used to select a colour.

name - The name of parameter. Default is "colour".

values - List of possible values. Default is all MAGICS colours.

default - The initial value. Default is "red". *Note that the colour names must be provided in lower-case letters.*

The following piece of code defines a colour field with three choices (red, blue or green), blue being the default :

```
a = colour(
    name      : "positive colour",
    values    : [ "RED", "BLUE", "GREEN" ],
    default   : "BLUE"
)
```



icon(...)

Creates an icon widget. This is used to accept an input icon dropped by the user. You can restrict the icons that are accepted to a given class (e.g. only contours). The list of arguments is :

name - The name of the parameter. Default is `icon`.

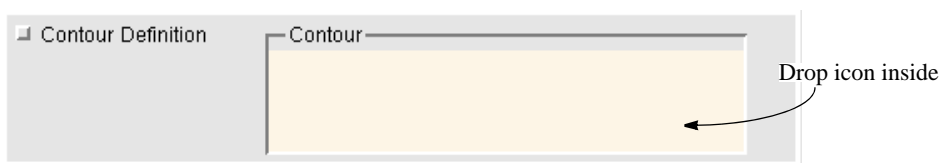
class - List of possible icon classes that can be dropped. Retrieve will accept only the Mars Retrieval icons. PCONT will accept Contour, POBS will accept BUFR Filter. GRIB will accept all the icons that produce GRIB output, i.e Mars Retrieval, Formula, GRIB Filter. Default is `anything`.

values - List of possible values. Default is `anything`.

default - The initial value. No default.

The following piece of code defines an ield for contour icons, with no default icon being defined :

```
a = icon(
    name : "contour definition",
    class : "PCONT"
)
```



option_menu(...)

Creates an option menu widget. This is used to choose a single value from a list of possible values.

name - Name of parameter. Default is `option_menu`.

values - List of possible values. Default is ["a", "b", "c"].

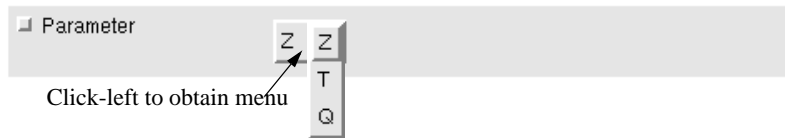
default - The initial value. Default is "a".

The following code defines a 3 choice option menu (Z, T or Q), Z being the default :

```

a = option_menu(
    name      : "parameter",
    values    : ["z", "t", "q"],
    default   : "z"
)

```



slider(...)

Creates a slider widget, to select from a range of integral values. The list of arguments is :

name - The name of the parameter. Default is "slider".

default - The initial value. Default is 0.

min - The minimum value. Default is 0.

max - The maximum value. Default is 100.

step - The increment value. Default is 10.

direction - The direction of the slide. Either "max_on_left" or "max_on_right" (default)

The following piece of code defines a slider for an input parameter varying between 1 and 10, with a default of 5 :

```

a = slider(
    name      : "number of 24h steps",
    min       : 1,
    max       : 10,
    default   : 5
)

```



toggle(...)

Creates an on/off widget, to activate or deactivate a setting. The list of arguments is :

name - The name of the parameter. Default is "toggle".

default - The initial value. Default is "on".

Using the *dialog()* Function

The `dialog()` function provides a straightforward way to code a user interface. Using this function to build the user interface, everything is self contained within a single macro program. The drawback is that the user interface is volatile, i.e. it is rebuilt everytime you run the macro. As a consequence, any values you may input on the user interface are not kept from one run to another.

If you prefer to keep you input values in the macro user interface you have to follow another approach using the complementary Macro Parameters icon (see "Using Macro Parameters" on page II- 110).

The way a macro user interface is coded with `dialog()` is as follows :

- you set up a number of input element functions to specify the user interface components
- collect the definition variables returned by the input element functions in a list and pass the list as input to the `dialog()` function
- `dialog()` returns a definition; the names of the members of this definition are given by the string specified in the `name` parameter of each input element function
- the input to the macro is extracted from the definition returned by `dialog()`

The following simplistic example makes this clear.

```
# setting up input element functions
a = colour(
    name : "contour colour"
)

b = icon(
    name : "data to plot",
    class : "GRIB"
)

# list of input element variables passed to dialog()
input = dialog([a,b])

# input retrieved from the definition returned by dialog()
field      = input["data to plot"]
colour     = input["contour colour"]

# now use it
plot_contour = pcont(
    contour_line_colour      : colour,
    contour_highlight_colour : colour
)
plot(field,plot_contour)
```

When you execute the macro, the user interface pops up. You specify the required input, click OK for acceptance and the program runs with the parameters you specified.

Using Macro Parameters

The other way to obtain a user interface uses an auxiliary module to Macro, named MacroParameters. You prepare a MacroParameters icon, which builds the user interface and passes the user input to the macro program for execution. The macro icon is not executable by itself.

In order to receive parameters from the MacroParameters icon, your macro program must implement a special handler named `edit` - see "Handlers" on page II- 53 and "Macro Run-Modes" on page II- 99. Within this handler, you place the input element functions and return their output as a list.

Then you implement other handlers, one for each action from the icon menu that you want to cater for (*Execute, Visualise, ...*). These handlers take the output from the `edit` handler as their input. Inside each handler you extract the input parameters and carry out whatever action you require. The same simplistic example presented above using the `dialog()` function is shown below using Macro Parameters.

```

mode = runmode()
if not(mode = "edit") then
    error = "Error : This macro must be called with parameters"
    fail (error)
end if

on edit
    a = colour(
        name : "contour colour"
    )
    b = icon(
        name : "data to plot",
        class: "GRIB"
    )
    return[a,b]
end edit

on visualise(in_list)
    field = in_list["data to plot"]
    colour = in_list["contour colour"]
    plot_contour = pcont(
        contour_line_colour    : colour,
        contour_highlight_colour : colour
    )
    plot(field,plot_contour)
end visualise

on prepare(in_list)
    field = in_list["data to plot"]
    colour = in_list["contour colour"]
    plot_contour = pcont(
        contour_line_colour    : colour,
        contour_highlight_colour : colour
    )
    return[field,plot_contour]
end prepare

```

To use this macro you have to drop it into the field of a MacroParameters editor window. This causes the user interface defined according to the input element functions inside the `edit` handler to be built within the editor window of MacroParameters.

Once you have input suitable values, click **Apply** for acceptance. Then pick an action from the MacroParameters icon menu (consistent with the handlers you have defined). The input you specify in the user interface remains saved within the MacroParameters icon.

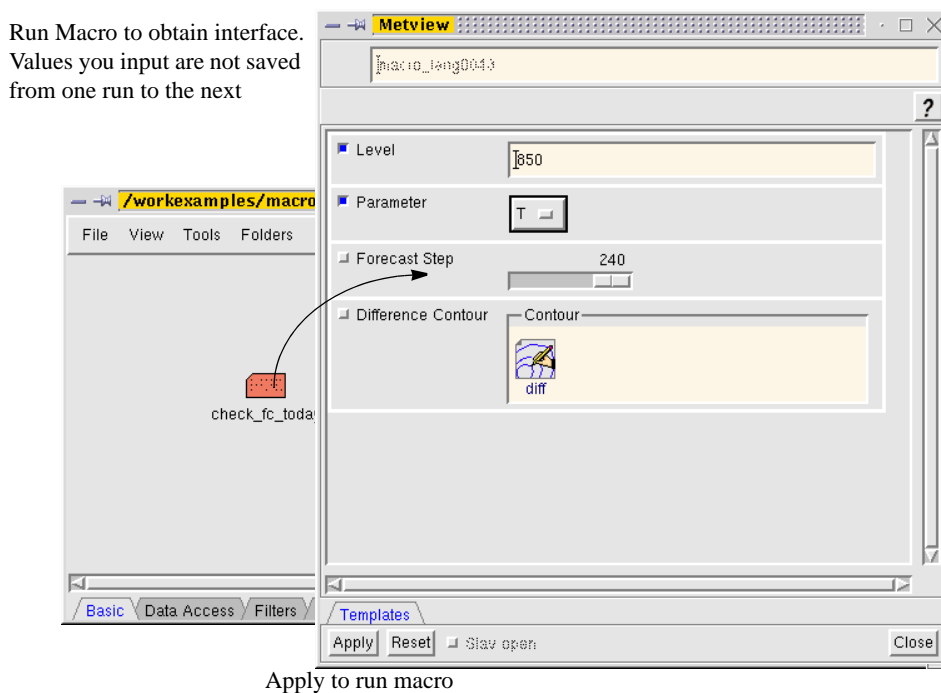
dialog() or Macro Parameters?

When should you choose to implement a user interface through the `dialog()` function rather than a MacroParameters icon? This depends on the exact usage you intend. The crucial difference is in the volatility of the resulting user interface.

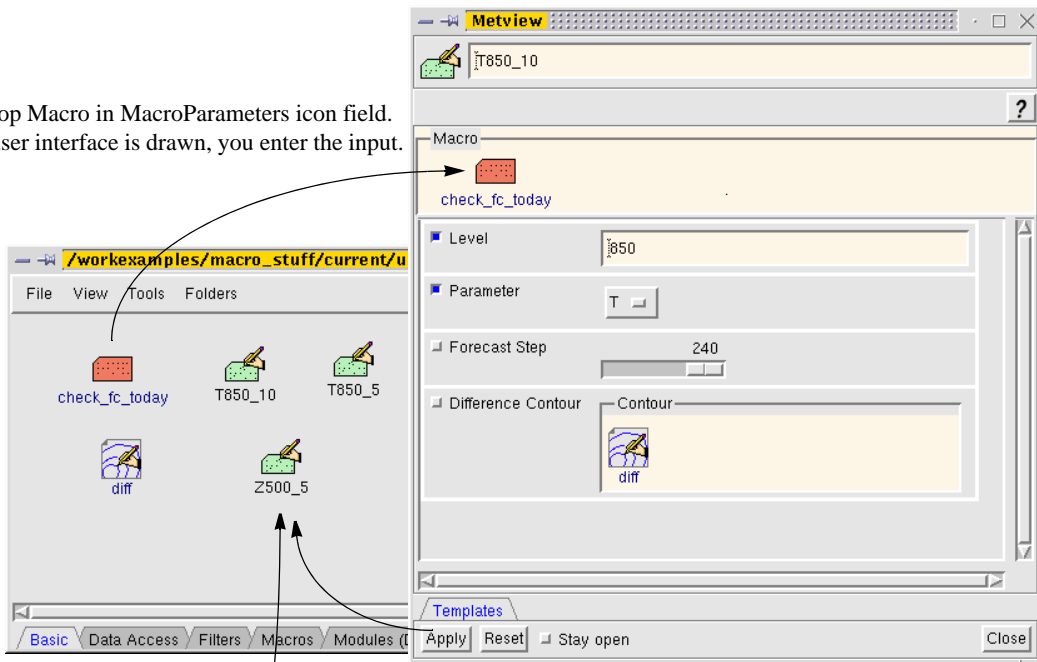
Using a macro icon with a `dialog()` function means that everytime you run the macro you have to enter the required values, unless you happen to be happy with the default values.

Using a MacroParameters icon to set up a user interface defined by the accompanying macro icon allows you to reuse the same interface. Specifically, since it is the macro icon that defines the user interface you can have any number of MacroParameter icons all using this same macro program; each can store a different set of input parameters and be used for predefined tasks. Any changes to the macro icon will affect all the MacroParameter icons.

Figure II-8 provides a pictorial representation of the operational differences between the two macro user interface implementations.



1) Drop Macro in MacroParameters icon field.
The user interface is drawn, you enter the input.



3) To obtain the required output, run the MacroParameters icon.

You cannot run the macro icon itself

2) Save MacroParameters icon

You can have as many MacroParameter icons as you require.
Each will store a different set of input parameters

Figure II-8 : Operational differences between the two user interface implementations - via a stand alone Macro using dialog() function (top) and through a Macro/MacroParameters combination (bottom). See text for details

METVIEW IN BATCH MODE

Metview can run macro programs in non-interactive batch mode, i.e. from the UNIX command line without recourse to the User Interface. This is used to run operational jobs coded as macro programs, as the running of these jobs can be set up in shell scripts and managed with scheduling software (i.e. run during periods of minimum activity and/or network traffic).

Running macros in batch presents no special requirements, except that display output is not allowed. Apart from this, the same macro can run in batch or interactively. The actions that a macro run in batch can carry out are :

- saving output to a file (GRIB, BUFR, ASCII, Geopoints) and / or
- produce a visualisation to a file (PS, JPEG, PNG) or printer (see "Output Formats and Destinations" on page II- 73 for details on how to code these outcomes).

You can make your macro handle several different usages by employing the `runmode()` function - this allows you to know in which mode the macro is being run and assign a suitable procedure to perform including which parameters should be set (e.g. output device).

Running Macro Programs in Batch

To run a macro program in batch, start Metview in batch mode by specifying the `-b` option, followed by the name of the macro program you want to run :

```
% metview -b the_macro
```

Frequently a line like this is part of a shell script, with lines preceding the call to Metview (e.g. copying data from a storage space, setting environment variables) and also following the call, e.g. to move output to another location or to send a generated PS file to a printer.

Passing Arguments in Batch

You can pass arguments to the macro running in batch directly from the command line. These are recovered within the macro and used as needed. To pass the arguments, just specify them after the macro name, i.e.

```
% metview -b the_macro Z 500 20010425
```

To recover the arguments, you need to use the `arguments()` function. This function returns a list of the arguments that you passed to the program, allowing you to retrieve them one by one. A simplistic program using the above command could be :

```
# function returns a list with the input arguments
```

```

x = arguments()
if count(x) <> 3 then
    fail("wrong number of args - needs Param, Level and date")
else
    # gets the input from the list
    param = x[1]
    level = x[2]
    date = date(x[3])
end if

# do something with the input
data = retrieve(
    param      : param,
    levelist   : level,
    date       : date,
    grid       : [2,2]
)
name = param & level & x[3] & "_grib"
write(name, an_1)

```

Note the safety net to catch errors specifying the input arguments. You could have added checks on the type of the arguments for further security.

A Batch Example

Here we present an example of a macro which can be used in two ways :

- in batch - produces a PS file
- visualised - produces on screen display window

by using `runmode()` to specify the correct action. Macros written in this way offer maximum flexibility.

```

# Starts by setting up two output definitions, one a ps file
# the other on screen display
ps_file = output(
    format      : "postscript",
    destination : "file",
    file_name    : "precip.ps"
    printer     : "ps_oa"
)

screen = output (
    format      : "screen"
)

# Checks run mode and acts accordingly
mode = runmode()

if mode = "batch" then

    # Get date from environment variable or use yesterday
    the_date = getenv("PLOT_DATE")

```

```
        if the_date = "" then
            the_date = -1
        end if

        setoutput(ps_file)

    else if mode = "visualise" then
        the_date = -1
        setoutput(screen)
    else
        fail("run mode not authorized - batch or visualise only")
    end if

    # Defines and carries out data retrievals
    in_common = (
        LEVTYPE      : "SFC",
        LEVELIST     : "OFF",
        DATE         : the_date,
        STEP         : 240,
        TYPE         : "FC",
        REPRES       : "GG"
    )

    CP_FC = retrieve(
        in_common,
        PARAM       : "CP"
    )

    LSP_FC = retrieve(
        in_common,
        PARAM       : "LSP"
    )

    # Adds precipis to obtain total 10 daily precipitation
    R240 = CP_FC + LSP_FC

    # Defines Visual Definition
    lev_list = [10,25,50,75,100,150,200,250,1000,2500]
    R240_contour = pcont(
        contour              : "off",
        contour_label        : "off",
        contour_min_level    : 10,
        contour_level_selection_type : "level_list",
        contour_level_list   : lev_list,
        contour_hilo         : "off",
        contour_shade        : "on",
        contour_shade_min_level : 10.0,
        contour_shade_min_level_density : 50.0,
        contour_shade_max_level_density : 1000.0,
        contour_shade_min_level_colour : "cyan",
        contour_shade_max_level_colour : "red",
        grib_scaling_of_derived_fields:"on"
    )

    # Plotting instruction
    dw = plot_superpage() # necessary in batch mode
    plot (dw, R240, R240_contour)
```

This macro can be run in batch either from the command line or from within a shell script. A script suitable for a Korn shell is :

```
PLOT_DATE=2001-06-21
export PLOT_DATE
metview -b precip10d
lpr -Pps_oa_c precip.ps
mv precip.ps /home/xyz/xy/my_store/store
```

This shell script runs the macro for the date 21/06/2001, prints the resulting PS file visualisation and moves it to a storage location.

PERFORMANCE AND TIMING

Metview includes a number of functions for timing parts of your macro. This should help find any bottlenecks and allow you to concentrate your optimisation efforts on the right places.

Stopwatch Functions

The stopwatch functions in the Macro language provide a method of timing parts of your macro. Only one stopwatch exists at a time, but it can provide multiple sub-timings (called 'laptimes') as well as an overall timing.

The stopwatch is started with the command `stopwatch_start(string)`, where `string` is a text string descriptive of what is being timed. This string, along with the current date and time will be printed to the macro's output. The stopwatch is stopped with the command `stopwatch_stop()`; the total time (user and system CPU, and wall clock) since the stopwatch was started is printed. Another function, `stopwatch_reset(string)`, is provided as a convenient way of stopping the current stopwatch and starting a new one.

The function `stopwatch_laptime(string)` provides a way to print sub-times without stopping the current stopwatch. The `string` argument is used in the resulting printout.

See also "Timing Functions" on page II- 189 for a list of the available functions.

Using the Stopwatch with MARS Retrievals

Note that for MARS requests the Macro interpreter starts the request and continues running the macro until the result of the MARS retrieve (variable) is actually used. This approach can improve efficiency, but also means that a little more effort is required in order to obtain accurate stopwatch timings for a MARS request. Thus, to time MARS requests you have to force the request to be finished before `stopwatch_laptime()` or `stopwatch_stop()` is called. This can be done by placing a `print()` command using the resulting variable just after the retrieval - this forces the Macro interpreter to wait until the variable is available before proceeding to the stopwatch function.

Example of Using the Stopwatch Functions

First the MARS requests are timed using stopwatch named 'retrieves' and then stopwatch is reset and given a new name 'compute'. Different laptime calls are given unique parameters to make it easy to locate them from the output stream:

An example of using these functions follows, with the parts of the macro being replaced with comments or '...' for brevity. In this example, we time two main sections of the code: the first is a set of data retrievals, the second performs computations with this data. Thus we use two stopwatches in sequence, each with some sub-timings in order to better hone in on any bottleneck.

```
stopwatch_start("retrieves")
```

```
q = retrieve (...)  
print(q)  
stopwatch_laptime("retrieve q")  
  
u= retrieve (...)  
print(u)  
stopwatch_laptime("retrieve u")  
  
v= retrieve (...)  
print(v)  
stopwatch_laptime("retrieve v")  
  
stopwatch_reset("compute")  
# compiled fortran code runs here ...  
stopwatch_laptime("fortran")  
  
# macro computations here ...  
stopwatch_laptime("advection")  
  
stopwatch_stop()
```

The output from the stopwatch commands in this macro follows:

```
[retrieves\start stopwatch] Mon Jun  4 17:02:01 2007  
...  
[retrieves\laptime] retrieve q: 0.05u/0s CPU [0 sec wall clock]  
...  
[retrieves\laptime] retrieve u: 0u/0s CPU [1 sec wall clock]  
...  
[retrieves\laptime] retrieve v: 0u/0s CPU [0 sec wall clock]  
[retrieves\reset - total time until now] 0.05u/0s CPU [1 sec wall clock]  
[retrieves\reset - start new stopwatch!] compute - Mon Jun  4 17:02:02  
2007  
...  
[compute\laptime] fortran: 0u/0s CPU [1 sec wall clock]  
[compute\laptime] advection: 0.06u/0s CPU [0 sec wall clock]  
[compute\laptime] plot: 0.01u/0s CPU [0 sec wall clock]  
[compute\stop - final total] 0.07u/0s CPU [1 sec wall clock]  
[compute\stop-watchRemoved!] Mon Jun  4 17:02:03 2007
```

LIST OF OPERATORS AND FUNCTIONS

The remainder of this manual provides a reference guide for the operators and functions available in the Metview Macro language, detailing their function, input and output types.

Operators and functions are organised by the variable type they conceptually relate to (which may not be the same type as they return).

The notation used conveys information about the usage of operators and functions - Operators and functions are represented in bold font with input and output types in normal font. Output type designation precedes an expression enclosed in brackets containing an operator action or a function declaration (name followed by input arguments type). The following examples make the notation clear :

output_type (**op** input_type)

An operator **op** acting on an expression of type input_type returns a quantity of type output_type, e.g.

fieldset (not fieldset)

means : Operator **not** acting on a fieldset results in another fieldset

output_type (input_type_1 **op** input_type_2)

An operator **op** acting on two expressions of type input_type_1 and input_type_2 returns a quantity of type output_type, e.g.

date (date + number)

means : Operator **+** acting on a date and a number (adding a number to a date) results in a date

output_type **function**(input_type_1,..., input_type_n)

A function **function** acting on n arguments of type input_type_1 to input_type_n,... returns a quantity of type output_type, e.g.

date date(number)

means : Function **date()** accepts a number as its single argument and returns a date.

Information Functions

Here are listed a few functions which convey information about expressions and other macro functions.

string *type*(any)

Returns the type of an expression as a string.

list *arguments*()

Returns the list of the arguments with which the current function was called. Used to retrieve the arguments passed to functions which are declared without an argument list (see "Declaring functions" in "Macro Function Essentials" on page II- 44) and to retrieve arguments passed to a macro program run in batch mode.

none *describe*(string)

Prints a terse one-line description of the function whose name is passed as the argument.

list *dictionary*()

Returns a list of all the available functions.

The nil Operand

The `nil` value defined in the Metview Macro language can be used in the ways described below - an example of the use of `nil` for initialisation of lists can be found in "Lists" on page II- 16.

any (`nil & any`)
any (`any & nil`)

The value `nil` can be concatenated to any other value. The result is the original value.

number (`nil = any`)
number (`any = nil`)
number (`nil <> any`)
number (`any <> nil`)

Any value can be compared with `nil`. Only `nil` is equal to `nil`. Returns a number, either 0 (false) or 1 (true)

number *count* (`nil`)

Returns 0. Note that in versions of Metview prior to version 3.9.3 returned 1.

Functions and Operators on Numbers

No distinction is made between integer or real numbers. All numbers are internally coded as double precision floating point reals

number (number *op* number)

Operation between two numbers. *op* is one of the operators below. These either return a number :

+	Addition	-	Subtraction
*	Multiplication	/	Division
^	Power		

or return 1 (result is true) or 0 (result is false) :

>	Larger Than	<	Smaller Than
>=	Larger or Equal	<=	Smaller or Equal
=	Equal	<>	Not Equal

number (number *and* number)

number (number *or* number)

number (*not* number)

Conjunction, Disjunction and Negation. Boolean operators consider all null values to be false and all non null values to be true. Result is either 1 or 0.

number *abs*(number)

Returns absolute value of a number

number *acos*(number)

number *asin*(number)

number *atan*(number)

Return the arc trigonometric function of a number. Result is in radians

number *cos*(number)

Return the cosine of a number (angle in radians)

number *exp*(number)

Returns the exponential of a number

number *int*(number)

Returns integer part of a number (no rounding, e.g. `int(1.999)=1.0`)

number *intbits*(number,number)

number *intbits*(number,number,number)

Takes the integer part of the first number and extracts a specified bit (or number of bits if a third number parameter is specified), where bit number 1 is the least significant bit (lsb). A single bit will always be returned as 1 or 0, regardless of its position in the integer. A group of bits will be treated as if the first bit is the least significant bit of the result. A few examples illustrate.

To extract the 1st, 2nd and 3rd bits from a number separately:

```
n = 6 # in bit-form, this is '00000110' with the lsb at the right
flag = intbits (n, 1) # flag is now 0
flag = intbits (n, 2) # flag is now 1
flag = intbits (n, 3) # flag is now 1
```

To extract the 1st and 2nd bits together to make a single number:

```
flag = intbits (n, 1, 2) # flag is now 2
```

To extract the 2nd and 3rd bits together to make a single number:

```
flag = intbits (n, 2, 2) # flag is now 3
```

To extract the 3rd and 4th bits together to make a single number:

```
flag = intbits (n, 3, 2) # flag is now 1
```

The number of bits available depends on the machine architecture and Metview's compilation options, but at the time of writing it should be 32.

number *log*(number)

Returns the natural logarithm of a number

number *log10*(number)

Returns the logarithm base 10 of a number

number *max*(number,number,...)

number *min*(number,number,...)

Returns maximum / minimum of the input values

number *mod*(number,number)

Returns the remainder of the division of the first value by the second. If the second number is larger than the first, it returns the integer part of the first number. Note that only the integer parts of the inputs are considered in the calculation, meaning that a second parameter of 0.5 would cause a division by zero.

number *neg*(number)

Returns the negative of a number. The same as (-number).

number *number*(date,string)

Converts a date to a number according to the number date format specified as the second input argument.

If date = 1997-04-01 02:03:04 (say), the available number date formats result in :

- YY gives 97
- YYYY gives 1997
- m or mm give 4
- d or dd give 1
- D or DDD give 91 (4th of April is the 91st day of the year).
- H or HH give 2
- M or MM give 3
- S or SS gives 4

number *precision*()**number *precision*(number)**

Sets the printing precision for floating point values, i.e. how many significant digits are used when printing or writing to a file. The value returned is the current precision value. Called with no arguments, it resets the precision to its default value, i.e. 12. Examples of printed output for `print(1234.56789)`:

```
precision( 12 ) gives 1234.56789
precision( 6 ) gives 1234.57
precision( 4 ) gives 1235
precision( 2 ) gives 1.2e+03
```

number *random*()

Returns a randomly selected non-negative double-precision floating-point value. The return values are uniformly distributed between [0.0, 1.0). There is no need to 'seed' this random function, as this is done automatically the first time it is called.

number *round*(number,number)

Rounds off spurious decimals in a value. The first number is the value to be rounded, the second is the number of decimal places to leave. Examples of values returned by `round(v,n)` for `v = 1234.56789`:

```
round( v, 1 ) gives 1234.6
round( v, 3 ) gives 1234.568
round( v, -2 ) gives 1200
```

number *sgn*(number)

Returns the sign of a number as a number : -1 for negative values, 1 for positive and 0 for null values.

number *sin*(number)

Return the sine of a number (angle in radians)

number *sqrt*(number)

Returns the square root of a number

string *string*(number)

Returns the string equivalent of a number

number *sum*(number,number,...)

Returns the sum of the input values

number *tan*(number)

Return the tangent of a number (angle in radians)

Functions and Operators on Strings

string (string *op* string)

Comparison between two strings, returning either 0 (false) or 1 (true); *op* is one of the boolean operators below :

>	Larger Than	<	Smaller Than
>=	Larger or Equal	<=	Smaller or Equal
=	Equal	<>	Not Equal

Smaller, greater, equal applied to strings refer to lower, higher, same alphabetical order. The comparison is case sensitive and is done using the ASCII code of each letter, hence the following expression is `true` (returns 1) :

```
"ABC" < "abc"
```

Type `man ascii` at the UNIX prompt to know more about the ASCII encoding of characters.

string (string & string & ...)
 string (string & number & ...)
 string (number & string & ...)
 string (string & date & ...)
 string (date & string & ...)

Returns a string equal to the concatenation of strings or strings and numbers and/or dates. If concatenating a date, the date is first converted to a string using the default string date format (see "Converting date components" on page II- 13).

string *ascii*(number)

Returns a string consisting of one character whose ASCII code has been provided as the single parameter to the function. For example:

```
linefeed = ascii (10)
```

number *length*(string)

This function returns the length of a string

```
word = "hello"  
print (word " is ", length(word), "characters long")
```

string *lowercase*(string)

Returns a lowercase copy of the input string.

list *parse*(string)

list *parse*(string,string)

This function splits the first input string at each occurrence of any of the field separators specified as the second string. It returns a list whose elements are the split tokens of the input string.

Macro assigns a type to each of these components (i.e. number or string). Space (" ") is the default separator when none is specified by the user, but any combination of characters can be specified as the set of separators.

```
# specify a comma and space as separator
s = "test1, 512.0, 498.0, 10.0"
f = parse(s, ", ")

# now access each retrieved element by indexing the list
print ("result of ", f[1], " : ", (f[2]-f[3])/f[4])
```

this prints :

```
result of test1 : 1.4
```

Supplying an empty string as the second parameter causes a complete list of the string's characters to be returned. For example :

```
parse ("Metview", "")
```

returns a list :

```
[M,e,t,v,i,e,w]
```

The `parse()` function is useful to parse text input when reading ASCII files within a macro program (see "General ASCII input" on page II- 34).

string *search* (string,string)

Searches the first string for the second string. The return value is the index of the first occurrence of the second string in the first. If the search fails, then it returns -1. Note that the comparison is case-sensitive.

For example :

```
filename = 'z_t2m_u_v_20060717.grib'
t2m_index = search (filename, 't2m')
```

returns the value 3.

string *substring* (string,number,number)

Returns a substring of the input string. The second parameter specifies the index of the first character to be retrieved (1 is the first character). The third parameter specifies the index of the last character to be retrieved. For example :

```
substring ("Metview", 2, 4)
```

returns the string "etv".

string *string*(date,string)

Converts a date to a string according to the string date format specified as the second input argument.

If `date = 1997-04-01 02:03:04` (say), the available string date formats result in :

- `YY` gives 97
- `YYYY` gives 1997
- `m` gives 4
- `mm` gives 04
- `mmm` gives Apr
- `mmm` gives April
- `d` gives 1
- `dd` gives 01
- `ddd` gives Tue
- `dddd` gives Tuesday
- `D` gives 91 (4th of April = julian day 91; 92 for a leap year).
- `DDD` gives 091
- `H` gives 2
- `HH` gives 02
- `M` gives 3
- `MM` gives 03
- `S` gives 4
- `SS` gives 04
- Any other character is copied as such.

string *uppercase*(string)

Returns an uppercase copy of the input string.

Functions and Operators on Dates

number (date - date)

Returns the number of days between two dates.

date (date + number)

date (date - number)

Adds/subtracts a number of days to/from a date. Returns a date.

number (date *op* date)

Comparison between two dates, returning either 0 (false) or 1 (true); *op* is one of the boolean operators below :

>	Larger Than	<	Smaller Than
>=	Larger or Equal	<=	Smaller or Equal
=	Equal	<>	Not Equal

Smaller, greater, equal applied to dates refer to earlier, later, same dates.

date *addmonths*(date, number)

Returns a date whose value is the input date plus the number of months specified as the second argument.

date *date*(number)

Creates a date from a number. If the number is negative or zero, the parameter is the number of days from the current day. Otherwise, the number must represent a date in the *yyymmdd* or *yyyymmdd* formats. The hour, minute and second information of the output date is lost (set to 0). Use *hour()*, *minute()*, *second()* to specify/restore it.

number *day*(date)

number *dow*(date)

Returns respectively the day of the month / day of week (monday is 1, sunday is 7) part of a date.

number *hour*(number)

Converts a number of hours into a number of days which can then be added to a date, e.g. to provide hour information to a date created by the *date()* function. Equivalent to dividing by 24.

number *hour*(date)

Returns the hour part of a date.

number *julday* (date)

number *juldate*(date)

Returns a date as Julian day and Julian date, respectively.

number *minute*(number)

Converts a number of minutes into a number of days which can then be added to a date, e.g. to provide minute information to a date created by the `date()` function. Equivalent to dividing by 1440.

number *minute*(date)

Returns the minute part of a date.

number *month*(date)

Returns the month part of a date.

date *now*()

Creates a date from the current day and time.

number *number*(date,string)

Converts a date to a number according to the number date format specified as the second input argument. See the same entry in "Functions and Operators on Numbers" on page II- 125

number *second*(number)

Converts a number of seconds into a number of days which can then be added to a date, e.g. to provide seconds information to a date created by the `date()` function. Equivalent to dividing by 86400.

number *second*(date)

Returns the second part of a date.

string *string*(date,string)

Converts a date to a string according to the string date format specified as the second input argument. See the same entry in "Functions and Operators on Strings" on page II- 129

number *year*(date)

Returns the year part of a date.

number *yymmdd*(date)

Returns a date as a 6 digit number - discards hours, minutes and seconds.

number *yyyymmdd*(date)

Returns a date as an 8 digit number - discards hours, minutes and seconds.

Functions and Operators on Lists

any `list[number]`

list `list[number,number]`

list `list[number,number,number]`

Apply the `[]` to a list to return element(s) from that list. Note that array indices start at 1 :

- `mylist[n]` returns the *n*th element of `mylist`
- `mylist[n,m]` returns the *n*th to the *m*th elements of `mylist`
- `mylist[n,m,i]` returns every *i*th of the *n*th to the *m*th elements of `mylist`

```
# copies elements 1, 5, 9, 13, 17 of x into y
Y = X[1,20,4]
```

number (any *in* list)

number (any *not in* list)

Tests whether a value is in a list or not. Returns a 0 (false) or 1 (true)

list (list & list)

Concatenate two lists.

number *count*(list)

Returns the number of elements in a list.

list *list*(any,any,...)

Returns a list built from its arguments.

list *sort*(list)

Sorts a list in ascending order.

list *sort*(list,string)

Sorts a list given a comparison, expressed as a string : Ascending "<", descending ">"; you may specify the sorting criterium in a comparison function :

```
function compare(a,b)
  return a < b
end compare

numbers = [1,5,3,9,0,4,6,7,8,2]

print (sort(numbers, ">")) # prints in decreasing order
print (sort(numbers, "compare"))# prints in ascending order
```

Note that it is not valid to sort a list which contains more than one type of data element.

```
list sort_indices( list )
list sort_indices( list,string )
```

Sorts a list and returns the sorted indices. The default behaviour is to sort in ascending order unless an alternative comparison function is provided. See example under `sort_and_indices()` to see how this function works.

```
list sort_and_indices( list)
list sort_and_indices( list,string )
```

Sorts a list and returns a list of pairs of list items and their corresponding indices in the original list. The default behaviour is to sort in ascending order unless an alternative comparison function is provided. The following example illustrates `sort_indices()` and `sort_and_indices()`:

```
original      = [10, 12, 9, 7, 6]
comparison    = "<"

sorted_list   = sort          (original, comparison)
sorted_indices = sort_indices (original, comparison)
sorted_both   = sort_and_indices (original, comparison)

print ('Original list   : ', original)
print ('Sorted list     : ', sorted_list)
print ('Sorted indices  : ', sorted_indices)
print ('Sort and indices : ', sorted_both)
```

Note that in this example it is not necessary to provide a comparison operator, as "<" is the default anyway. The output is as follows:

```
Original list   : [10,12,9,7,6]
Sorted list     : [6,7,9,10,12]
Sorted indices  : [5,4,3,1,2]
Sort and indices : [[6,5],[7,4],[9,3],[10,1],[12,2]]
```

Functions and Operators on Fieldsets

See first "How operators and functions work on fieldsets" on page II- 19.

fieldset (fieldset *op* fieldset)

Operation between two fieldsets. *op* is one of the operators below :

+	Addition	-	Subtraction
*	Multiplication	/	Division
^	Power		

The fieldsets returned by these boolean operators are boolean fieldsets (containing only 1 where result is true, 0 where it is false) :

>	Larger Than	<	Smaller Than
>=	Larger or Equal	<=	Smaller or Equal
=	Equal	<>	Not Equal

In all of the above operations, a missing value in one of the input fieldsets results in a corresponding missing value in the output fieldset.

fieldset (fieldset *op* number)

fieldset (number *op* fieldset)

Operations between fieldsets and numbers. *op* is any of the operations defined above. A missing value in either input fieldset will result in a missing value in the corresponding place in the output fieldset.

geopoints (fieldset *op* geopoints)

geopoints (geopoints *op* fieldset)

Operations between fieldsets and geopoints. *op* is any of the operations defined above. Missing values, both in the fieldset and in the original geopoints variable result in a value of `geo_missing_value`.

fieldset (fieldset *and* fieldset)

fieldset (fieldset *or* fieldset)

fieldset (*not* fieldset)

Conjunction, Disjunction and Negation. Boolean operators consider all null values to be false and all non null values to be true. The fieldsets created by boolean operators are binary fieldsets (containing only 1 where result is true, 0 where it is false). For example :

```
a = retrieve(...)
b = retrieve(...)
c = a and b
```

creates a fieldset *c* with values of 1 where the corresponding values of fieldset *a* and fieldset *b* are both non zero, and 0 otherwise. For an example of the use of boolean operators, see "fieldset mask(fieldset,list)" on page II- 151. A missing value in either input fieldset will result in a missing value in the corresponding place in the output fieldset.

fieldset (fieldset **and** number)
fieldset (number **or** fieldset)

Boolean operations between fieldsets and numbers. See above. A missing value in either input fieldset will result in a missing value in the corresponding place in the output fieldset.

geopoints (fieldset **and** geopoints)
geopoints (geopoints **or** fieldset)

Boolean operations between fieldsets and geopoints. See above.

fieldset (fieldset **&** fieldset **&** ...)
fieldset (nil **&** fieldset **&** ...)
fieldset (fieldset **&** nil)
fieldset **merge**(fieldset,fieldset,...)

Merge several fieldsets. The output is a fieldset with as many fields as the total number of fields in all merged fieldsets. Merging with the value `nil` does nothing, and is used to initialise when building a fieldset from nothing.

fieldset fieldset[number]
fieldset fieldset[number,number]
fieldset fieldset[number,number,number]

Extract a selection of fields from a fieldset. If one parameter is given, only one field is selected. If two parameters are given, the fields ranging from the first to the last index are returned. The optional third parameter represents an increment *n* - every *n*th field from the first to the last index are returned.

```
# copies fields 1, 5, 9, 13, 17 of x into y
Y = X[1,20,4]
```

fieldset **abs**(fieldset)

Returns the fieldset of the absolute value of the input fieldset at each grid point or spectral coefficient. Missing values are retained, unaltered by the calculation.

fieldset **acos**(fieldset)
fieldset **asin**(fieldset)
fieldset **atan**(fieldset)

Return the fieldset of the arc trigonometric function of the input fieldset at each grid point. Result is in radians. Missing values are retained, unaltered by the calculation.

fieldset *cos*(fieldset)

Returns the fieldset of the cosine of the input fieldset at each grid point. Input values must be in radians. Missing values are retained, unaltered by the calculation.

number *count*(fieldset)

Returns the number of fields in a fieldset.

fieldset *exp*(fieldset)

Returns the fieldset of the exponential of the input fieldset at each grid point. Missing values are retained, unaltered by the calculation.

fieldset *float*(fieldset, number)

Returns a fieldset with integer data converted into floating point data for more accurate computations. The second parameter is optional; if given it defines the number of bits used for packing the float values. If not given, the default value of 24 is used (unless function `gribsetbits(number)` has been called to set it).

fieldset *int*(fieldset)

Returns the fieldset of the integer part of the input fieldset at each grid point or spectral coefficient. Missing values are retained, unaltered by the calculation.

fieldset *integer*(fieldset)

Returns the fieldset of the integer part of the input fieldset at each grid point or spectral coefficient. This function modifies the resulting GRIB header to be of integer data type, and is intended for the conversion of satellite data so that it can be plotted correctly by the MAGICCS library. Missing values are replaced with `INT_MAX` and should therefore continue to be interpreted as missing.

fieldset *log*(fieldset)

Returns the fieldset of the natural log of the input fieldset at each grid point. Missing values are retained, unaltered by the calculation.

fieldset *log10*(fieldset)

Returns the fieldset of the log base 10 of the input fieldset at each grid point. Missing values are retained, unaltered by the calculation.

fieldset *neg*(fieldset)

Returns the fieldset of the negative of the input fieldset at each grid point or spectral coefficient. The same as `(-fieldset)`. Missing values are retained, unaltered by the calculation.

fieldset *sgn*(fieldset)

Returns the fieldset of the sign of the values of the input fieldset at each grid point or spectral coefficient : -1 for negative values, 1 for positive and 0 for null values. Missing values are retained, unaltered by the calculation.

fieldset *sin*(fieldset)

Returns the fieldset of the sine of the input fieldset at each grid point. Input fieldset must have values in radians. Missing values are retained, unaltered by the calculation.

fieldset *sqrt*(fieldset)

Returns the fieldset of the square root of the input fieldset at each grid point. Missing values are retained, unaltered by the calculation.

fieldset *tan*(fieldset)

Return the tangent of the input fieldset at each grid point. Input fieldset must have values in radians. Missing values are retained, unaltered by the calculation.

number or list *accumulate*(fieldset)

For each field in the fieldset, the `accumulate()` function calculates the sum of all the values of the field. If there is only one field in the fieldset, a number is returned. Otherwise, a list of numbers is returned. Only non-missing values are considered in the calculation. If there are no valid values, the function returns the grib missing value indicator.

number or list *average*(fieldset)

For each field in the fieldset, the `average()` function calculates the average of all the field values. If there is only one field in the fieldset, a number is returned. Otherwise, a list of numbers is returned. Only non-missing values are considered in the calculation. If there are no valid values, the function returns the grib missing value indicator.

Note that the `average()` function does not return the physically correct average of the field but merely the average of all field values. To get the physically correct average value of a field, use the `integrate()` function (which employs a cosine latitude weighting)

list *average_ew*(fieldset,list,number)

The function `average_ew()` takes as parameters a fieldset, a list of four numbers that define an area ([N,W,S,E]) and a number that defines the output one-dimensional grid interval in degrees.

The function returns a list (if the input fieldset contains only one field) or a list of lists. The elements of the returned list(s) are means computed over rows of similar latitude using those grid points that fall inside the given area. Means are computed at intervals as specified in the third parameter. The output list size is thus independent of the grid interval in the input fieldset.

Each grid point value is weighted by the cosine of its latitude. Missing values are ignored. If a latitude belt contains no grid point values then a missing value indicator is returned.

Example:

```
ave = average_ew(fs, [60,-180,-60,180], 2.5)
```

This function call will compute means over full latitude circles starting from 60N, stepping 2.5 degrees until 60S. If *fs* contains only one field the output would be a list of 49 E-W mean values, from North to South. If *fs* contains *n* fields then the output would be a list of *n* lists, where each of these *n* sublists would contain 49 means.

For the above example, each value returned (representing the mean at latitude *Lat*) is the mean of non-missing values in those grid points whose latitude coordinate is between *Lat*-1.25 and *Lat*+1.25 (1.25 is 2.5/2), i.e. within a latitude belt with width of 2.5 degrees, centered around *Lat*.

list *average_ns*(fieldset,list,number)

The function *average_ns*() takes as parameters a fieldset, a list of four numbers that define an area ([N,W,S,E]) and a number that defines the output one-dimensional grid interval in degrees.

The function returns a list (if the input fieldset contains only one field) or a list of lists. The elements of the returned list(s) are means computed over lines of similar longitude using those grid points that fall inside the given area. Means are computed at intervals as specified in the third parameter. The output list size is thus independent of the grid interval in the input fieldset.

Each grid point value is weighted by the cosine of its latitude. Missing values are ignored. If a longitude line contains no grid point values then a missing value indicator is returned.

Example:

```
ave = average_ns(fs, [30,0,-30,360], 5)
```

This function call will compute means over longitudes 30N...30S, in 5 degree intervals around the globe. The result for each field in *fs* would be a list of 73 values (in this case values for 0 and 360 are duplicated values).

Each value returned (representing the mean at longitude *Lon*) is a mean of non-missing values in those grid points whose longitude coordinate is between *Lon*-2.5 and *Lon*+2.5 (2.5 is 5/2), in the belt between 30N and 30S.

fieldset *bitmap* (fieldset,number)

fieldset *bitmap* (fieldset,fieldset)

Returns a copy of the input fieldset (first argument) with zero or more of its values replaced with grib missing value indicators. If the second argument is a number, then any value equal to that number in the input fieldset is replaced with the missing value indicator. If the second argument is another fieldset with the same number of fields as the first fieldset, then the result takes the arrangement of missing values from the second fieldset. If the second argument is another fieldset with one field, the arrangement of missing values from that field are copied into all fields of the output fieldset. See also *nobitmap*.

number or list **corr_a**(fieldset,fieldset)
 number or list **corr_a**(fieldset,fieldset,list)

Computes the correlation between two fieldsets over a weighted area. The area, if specified, is a list of numbers representing North, West, South, East. If the area is not specified, the whole field will be used in the calculation. The result is a number for a single field, or a list for a multi-field fieldset.

Note that the following lines are equivalent, although the first is more efficient:

```
z = corr_a (x, y)
z = covar_a (x, y) / (sqrt(var_a(x)) * sqrt(var_a(y)))
```

fieldset **coslat**(fieldset)

For each field in the input fieldset, this function creates a field where each grid point has the value of the cosine of its latitude.

fieldset **covar**(fieldset,fieldset)

Computes the covariance of two fieldsets. With n fields in the input fieldsets, if x_{ik} and y_{ik} are the i th values of the k th field of each input field respectively and z_i is the i th value of the resulting field, the formula can be written :

$$z_i = \frac{1}{n} \sum_{k=1}^n x_i^k y_i^k - \frac{1}{n} \sum_{k=1}^n x_i^k \sum_{k=1}^n y_i^k$$

Note that the following lines are equivalent:

```
z = covar(x,y)
z = mean(x*y) - mean(x) * mean(y)
```

A missing value in either input fieldset will result in a missing value in the corresponding place in the output fieldset.

number or list **covar_a**(fieldset,fieldset)
 number or list **covar_a**(fieldset,fieldset,list)

Computes the covariance of two fieldsets over a weighted area. The area, if specified, is a list of numbers representing North, West, South, East. If the area is not specified, the whole field will be used in the calculation. The result is a number for a single field, or a list for a multi-field fieldset.

list **datainfo**(fieldset)

Returns a list of definitions - one for each field in the fieldset. Each definition provides the following members: the index of the field in the fieldset, the number of missing values, the number of values that are present and the proportion of each. The following example illustrates how to use the function.

```
fs = read (strGribFile)

listdefInfo = datainfo (fs)
```

```

loop defInfo in listdefInfo
  print ("Field index           : ", defInfo.index)
  print ("Number of values present : ", defInfo.number_present)
  print ("Number of values missing  : ", defInfo.number_missing)
  print ("Proportion values present  : ", defInfo.proportion_present)
  print ("Proportion values missing  : ", defInfo.proportion_missing)
end loop

```

fieldset *direction*(fieldset,fieldset)

Returns a fieldset with the value in each grid point being the direction computed from the given U and V fieldsets; the first input fieldset is assumed to be the East-West (U) component and the second the North-South (V) component. The resulting numbers are directions, in degrees clockwise from North, where a value of 0 represents a wind from the North and a value of 90 represents a wind from the East.

fieldset *distance*(fieldset,number,number)
 fieldset *distance*(fieldset,list)

Returns a fieldset with the value in each grid point being the distance in meters from the given geographical location. The location may be specified by supplying either two numbers (latitude and longitude respectively) or a 2-element list containing latitude and longitude in that order. The location should be specified in degrees.

fieldset *div*(fieldset,fieldset)

Returns a fieldset with as many fields as the input fieldsets; the grid points of the output fieldset are the integer part of the division of the first fieldset by the second fieldset (the function operating field by field).

With n fields in the input fieldsets, if x_{ik} , y_{ik} are the i th value of the k th input fieldsets and z_i is the i th value of the resulting field:

$$z_i^k = \text{div}(x_i^k, y_i^k)$$

A missing value in either input fieldset will result in a missing value in the corresponding place in the output fieldset.

fieldset *duplicate*(fieldset,number)

Returns a fieldset with the specified number of copies of the field in the input fieldset. The input fieldset must contain only one field.

list *find*(fieldset,number)
 list *find*(fieldset,number,list)
 list *find*(fieldset,number,field)

A filtering function that returns a list of locations (lat/long pairs), where the values of the input fieldset given as the first argument equal the value specified as the second argument. Missing values in the input field are not returned.

- if there is a third argument, and it is a list of four numbers (lat/long coordinates) defining a geographical area - [North, West, South, East], the function returns a list of locations *within that area* where the fieldset values equal the input value
- if there is a third argument, and it is a mask field, the function returns a list of locations *within the area defined by the mask* (ie, where the mask gridpoints are non-zero) where the fieldset values equal the input value.

list *find*(fieldset,list)

list *find*(fieldset,list,list)

list *find*(fieldset,list,field)

A filtering function that returns a list of locations (lat/long pairs), where the values of the input fieldset given as the first argument are within the interval [a, b] specified as the second argument (a two value list). Missing values in the input field are not returned.

- if there is a third argument, and it is a list of four numbers (lat/long coordinates) defining a geographical area - [North, West, South, East], returns a list of locations *within that area* where the field values are within the interval [a, b]
- if there is a third argument, and it is a mask field, returns a list of locations *within the area defined by the mask* (ie, where the mask gridpoints are non-zero) where the fieldset values are within the interval [a, b]

list *frequencies*(fieldset,list)

list *frequencies*(fieldset,list,list)

Counts the number of grid points whose values fall within a set of specified intervals. These intervals are given as the second argument - a list of values in ascending order, starting with the upper bound of the first interval, eg [0, 10, 20]. A third argument, if given, specifies a geographical area over which to consider values - [North, West, South, East]. Missing values in the input field are not included in the results.

If the input fieldset has just one field, then the result is a list of n+1 elements where n is the number of elements in the interval list. Using the above example, the output list could be described as follows:

- the first element is the number of values below 0
- the second element is the number of values in the range [0, 10)
- the third element is the number of values in the range [10, 20)
- the fourth element is the number of values above 20

If the input fieldset has more than one field, the result is a list of lists, one for each field. Note that this function accumulates its results between fields in a fieldset.

geopoints *gfind*(fieldset,number)

geopoints *gfind*(fieldset,number,number)

A filtering function that returns a geopoints holding the grid points whose value is equal to the *value* of the first number. Missing values in the input field are not returned. If a second number is given as the third argument it is a tolerance *threshold* and the geopoints will hold the grid points for which :

```
abs(data-value) <= threshold
```

number or list *getksec1*(fieldset,number)
 number or list *getksec2*(fieldset,number)
 number or list *getksec3*(fieldset,number)
 number or list *getksec4*(fieldset,number)

Note: as of Metview 3.11, these functions are deprecated - please use the alternatives, *grib_get_long*, *grib_get_double* and *grib_get_string* on page II- 148.

These functions return the integer element of the GRIB header sections 1 (the product definition section), 2 (the grid description section), 3 (the bitmap section) and 4 (the binary data section), specified by the second argument, from the fieldset given as the first argument.

If the input fieldset has a single field, it returns a number, otherwise it returns a list of numbers. A list of the GRIB section element numbers and their meaning can be found in :

```
http://www.ecmwf.int/publications/manuals/libraries/gribex/ksec1.html
http://www.ecmwf.int/publications/manuals/libraries/gribex/ksec2.html
http://www.ecmwf.int/publications/manuals/libraries/gribex/ksec3.html
http://www.ecmwf.int/publications/manuals/libraries/gribex/ksec4.html
```

Example :

```
rh = retrieve(
    levelist : [850, 500],
    param    : "r",
    type     : "fc",
    step     : 240
)
year = getksec1(rh,10)
step = getksec1(rh,16)
first_lat = getksec2(rh,4)/1000
```

number or list *getrsec2*(fieldset,number)
 number or list *getrsec3*(fieldset,number)
 number or list *getrsec4*(fieldset,number)

Note: as of Metview 3.11, these functions are deprecated - please use the alternatives, *grib_get_long*, *grib_get_double* and *grib_get_string* on page II- 148.

These functions return the real-valued element of the GRIB header sections 2 (the grid description section), 3 (the bitmap section) and 4 (the binary data section), specified by the second argument, from the fieldset given as the first argument.

If the input fieldset has a single field, it returns a number, otherwise it returns a list of numbers. A list of the GRIB section element numbers and their meaning can be found in :

```
http://www.ecmwf.int/publications/manuals/libraries/gribex/psec2.html
http://www.ecmwf.int/publications/manuals/libraries/gribex/psec3.html
http://www.ecmwf.int/publications/manuals/libraries/gribex/psec4.html
```

number or list ***grib_get_long***(fieldset, string)
 number or list ***grib_get_double***(fieldset, string)
 number or list ***grib_get_string***(fieldset, string)
 list ***grib_get_long_array***(fieldset, string)
 list ***grib_get_double_array***(fieldset, string)

These functions return information from the given fieldset's GRIB header. Available keys (to be passed as the second parameter) can be inspected by Examining the GRIB file (right-click, Examine). Alternatively, use the GRIB API command ***grib_dump*** to see the available key names. See http://www.ecmwf.int/publications/manuals/grib_api/keys.html for more details on key names.

The first three functions return a number if the input fieldset has a single field, otherwise they return a list of numbers. The 'array' functions return a list of numbers if the input fieldset has a single field, otherwise they return a list of lists of numbers.

The following example shows the retrieval of GRIB header information, including the derived key 'max', using the different functions:

```
print (grib_get_long (data, "editionNumber"))
print (grib_get_long (data, "max"))
print (grib_get_double (data, "max"))
print (grib_get_string (data, "max"))
print (grib_get_string (data, "typeOfGrid"))
```

The output from this on an example single-field GRIB file was:

```
1
317
317.278808594
317.279
regular_ll
```

The following example shows how to obtain the list of parallels from a reduced Gaussian grid fieldset:

```
a = read('/x/y/z/data_in_gg.grb')
pl = grib_get_long_array (a, 'pl')
print (count(pl))
print (pl)
```

As of Metview 3.11, these functions should be used in preference to the ***getksec*** and ***getrsec*** functions.

fieldset ***grib_set_long***(fieldset, list)
 fieldset ***grib_set_double***(fieldset, list)
 fieldset ***grib_set_string***(fieldset, list)

These functions set information in the given fieldset's GRIB header. The list provided as the second argument should be a set of key/value pairs, for example:

```
data = grib_set_long (data,
                    ["centre", 99,
                    "level", 200])
```

This function does not modify the input fieldset, but returns a new fieldset with the modifications applied.

Available keys can be inspected by Examining the GRIB file (right-click, Examine). Alternatively, use the GRIB API command *grib_dump* to see the available key names. See http://www.ecmwf.int/publications/manuals/grib_api/keys.html for more details on key names.

If applied to a multi-field fieldset, then all fields are modified.

As of Metview 3.11, these functions should be used in preference to the *putksec* and *putrsec* functions.

list ***gridvals***(fieldset)

This function returns the GRIB section 4 (i.e. the grid point values) as a list of numbers. If the fieldset contains more than one field it returns a list of lists. Each of these lists contains as many elements as grid points in each field. Missing values are included in the results.

```
# x is a fieldset of n fields
xgrid = gridvals(x)
field1_values = xgrid[1]
gridpoint1 = field1_values[1]

# or equivalently
gridpoint1 = xgrid[1][1]
```

number ***gribsetbits***(number)

This function sets the number of GRIB packing bits to the value given (eg 8, 10, 16), and returns the previously used internal value. This function is particularly useful when dealing with 10-bit satellite images as these require GRIB packing to be set to 10 bits.

number or list ***integrate***(fieldset)
 number or list ***integrate***(fieldset,list)
 number or list ***integrate***(fieldset,fieldset)

This function computes the average value of field. Each grid point value is weighted by the cosine of its latitude :

$$\frac{\sum x_i \cdot \cos \varphi_i}{\sum \cos \varphi_i}$$

Where φ_i is the latitude and x_i is the grid point value. If the input fieldset contains only one field, a number is returned. If there is more than one field, a list of numbers is returned. Missing values in the input fieldset are bypassed in this calculation. For each field for which there are no valid values, the grib missing indicator is returned.

- If the fieldset is the only argument, the integration is done on all grid points.
- If a list is the second argument, it must contain four numbers which are respectively the north, west, south and east boundaries of an area. The integration is done on the grid points contained inside this area :

```

europe = [75,-12.5,35,42.5]
x = integrate(field,europe)

```

- If a fieldset is the second argument it is used as a mask. It should contain either one or as many fields as the first fieldset. If it has a single field then this mask is applied to all fields of the input fieldset. If it has the same number of fields as the input fieldset, then a different mask is applied to each input field. The integration is performed only on the grid points where the mask values are non zero. The following code shows a simple example :

```

# Retrieve land-sea mask and interpolate to LL grid
lsm = retrieve(
    type      : "an",
    date      : -1,
    param     : "lsm",
    grid      : [1.5,1.5],
    levtype   : "sfc"
)

# The following line forces the values to 0 or 1.
lsm = lsm > 0.5

# Now compute the average value on land and on sea
land = integrate(field, lsm)
sea  = integrate(field, not lsm)

```

number or list *interpolate*(fieldset,list)

number or list *interpolate*(fieldset,number,number)

Interpolate a fieldset at a given point. If a list is given, it must contain two numbers - latitude and longitude. If two numbers are given, the first is the latitude, the second the longitude. The field must be a lat-long field. If the fieldset has only one field, a number is returned; otherwise a list is returned. Where it is not possible to generate a sensible value due to lack of valid data in the fieldset, the grib missing indicator is returned.

geopoints *interpolate*(fieldset,geopoints)

Generates a set of geopoints from a field. The first field of the input fieldset is used. The field is interpolated for each position of the geopoints given as a second parameter. The output geopoints take their date, time and level from the fieldset. Where it is not possible to generate a sensible value due to lack of valid data in the fieldset, the internal geopoints missing value is used (this value can be checked for with the built-in variable `geo_missing_value` or removed with the function `remove_missing_values`).

fieldset *lookup*(fieldset,fieldset)

fieldset *lookup*(fieldset,list)

These two functions build an output fieldset using the values in the first input fieldset as indices in a look-up action on a second input fieldset or input list :

- Takes the grid values in the first fieldset and uses them as index in the second fieldset. E.g. a grid value of `n` in the first fieldset, retrieves the corresponding grid point value of the $(n-1)$ th field of the second fieldset (indexing is 0 based). The output fieldset is built from these values and has as many fields as the second fieldset.

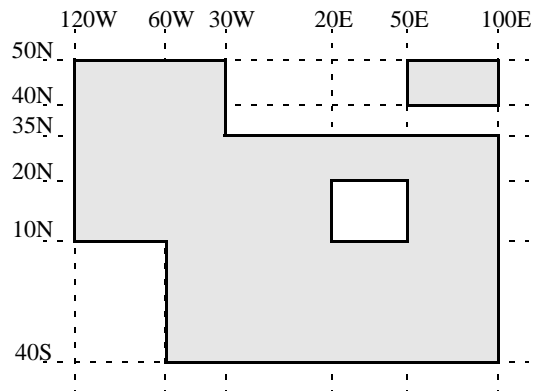
- Takes the grid values in the first fieldset and uses them as index in the list - real numbers are truncated, not rounded. E.g. a grid value of n in the first fieldset, retrieves the $(n-1)$ th list element (indexing is 0 based). The output fieldset is built from these values and has as many fields as the first fieldset.

Any missing values in the first fieldset will cause the function to fail with a 'value out of range' error message.

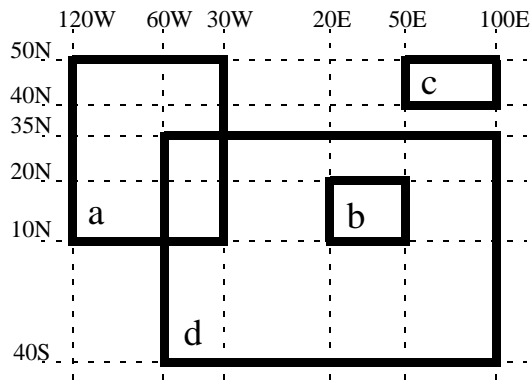
fieldset *mask*(fieldset,list)

For each field of the input fieldset, this function creates a field containing grid point values of 0 or 1 according to whether they are outside or inside a defined geographical area.

The list parameter must contain exactly four numbers representing a geographical area. These numbers should be in the order north, west, south and east (negative values for western and southern coordinates). Non-rectangular masks, and even convex masks can be created by using the operators *and*, *or* and *not*. To create the following mask :



First decompose into basic rectangles :



Then create a mask for each of them and use *and* and *or* to compose the desired mask. This is the corresponding macro :

```
# Define basic rectangles
a = [ 50,-120,10,-30 ]
b = [ 20,20,50,10 ]
c = [ 50,50,40,100 ]
d = [ 35,-60,-40,100 ]
```

```

# The field is used to get the grid information
f = retrieve(...)

# First compute the union of a,c and d
m = mask(f,a) or mask(f,d) or mask(f,c)

# Then remove b
m = m and not mask(f,b)

```

The resulting mask field can be used in the `integrate()` function.

fieldset **max**(fieldset)
fieldset **min**(fieldset)

Returns the fieldset of maximum (minimum) value of the input fieldset at each grid point or spectral coefficient. A missing value in either input fieldset will result in a missing value in the corresponding place in the output fieldset.

fieldset **max**(fieldset,fieldset)
fieldset **min**(fieldset,fieldset)

Returns the fieldset of maximum (minimum) value of the two input fieldsets at each grid point or spectral coefficient. A missing value in either input fieldset will result in a missing value in the corresponding place in the output fieldset.

fieldset **max**(fieldset,number)
fieldset **min**(fieldset,number)

Returns the fieldset of the maximum (minimum) of the number and the fieldset value at each grid point or spectral coefficient. Missing values in the input fieldset are transferred to the output fieldset.

geopoints **max**(fieldset,geopoints)
geopoints **min**(fieldset,geopoints)

Returns geopoints of maximum (minimum) of the fieldset value and the geopoint value at each grid point or spectral coefficient. Missing values, either in the fieldset or in the original geopoints variable, result in a value of `geo_missing_value`.

number **maxvalue**(fieldset)
number **maxvalue**(fieldset,list)
number **minvalue**(fieldset)
number **minvalue**(fieldset,list)

Returns the maximum (minimum) value of all the values of all the fields of the fieldset. The versions that take a list as a second parameter require a geographical area (north, west, south, east); only points within this area will be included in the calculation. Only non-missing values are con-

sidered in the calculation. If there are no valid values, the function returns the grib missing value indicator.

matrix or list *matrix*(fieldset)

Generates a matrix containing the values of the input field, or a list of matrices if there are more than one field in the fieldset. Only works with regular lat/long grids.

fieldset *mean*(fieldset)

Computes the mean field of a fieldset. A missing value in any field will result in a missing value in the corresponding place in the output fieldset. With n fields in the input fieldset, if x_{ik} is the i th value of the k th input field and y_i is the i th value of the resulting field, the formula can be written :

$$y_i = \frac{1}{n} \sum_{k=1}^n x_i^k$$

fieldset *mean_ew*(fieldset)

Takes a fieldset as its parameter and computes the mean for each line of constant latitude. The result is a fieldset where the value at each point is the mean of all the points at that latitude. Missing values are excluded; if there are no valid values, then the grib missing value indicator will be returned for those points.

fieldset *merge*(fieldset,fieldset,...)

Merge several fieldsets. The same as the operator $\&$. The output is a fieldset with as many fields as the total number of fields in all merged fieldsets. Merging with the value `nil` does nothing, and is used to initialise when building a fieldset from nothing.

fieldset *mod*(fieldset,fieldset)

Returns a fieldset with as many fields as the input fieldsets; the grid point values of the output fieldset are the remainder of the division of the first fieldset by the second fieldset (the function operating field by field). Where the gridpoint values of the second fieldset are larger than those of the first, the output gridpoint value is set to the integer part of the first input gridpoint value. A missing value in either input fieldset will result in a missing value in the corresponding place in the output fieldset. Note that only the integer parts of the inputs are considered in the calculation, meaning that a second parameter of 0.5 would cause a division by zero.

With n fields in the input fieldsets, if x_{ik} , y_{ik} are the i th value of the k th input fieldsets and z_i is the i th value of the resulting field:

$$z_i^k = \text{mod}(x_i^k, y_i^k)$$

number or list *nearest_gridpoint*(fieldset,list)
 number or list *nearest_gridpoint*(fieldset,number,number)

Returns the value of the nearest point to a given location in each field of a fieldset. If a list is given, it must contain two numbers - latitude and longitude. If two numbers are given, the first is the latitude, the second the longitude. The field must be a lat-long field. If the fieldset has only one field, a number is returned; otherwise a list is returned. Where it is not possible to generate a sensible value due to lack of valid data in the fieldset, the grib missing indicator is returned.

geopoints *nearest_gridpoint*(fieldset,geopoints)

Generates a set of geopoints from a field. The first field of the input fieldset is used. The result is a set of geopoints whose values are those of the nearest gridpoints in the field to the geopoints given as a second parameter. Where it is not possible to generate a sensible value due to lack of valid data in the fieldset, the internal geopoints missing value is used (this value can be checked for with the built-in variable `geo_missing_value` or removed with the function `remove_missing_values`).

list *nearest_gridpoint_info*(fieldset,list)
 list *nearest_gridpoint_info*(fieldset,number,number)

Returns the value and location of the nearest point to a given location in each field of a fieldset. If a list is given, it must contain two numbers - latitude and longitude. If two numbers are given, the first is the latitude, the second the longitude. The field must be a lat-long field. The return value is a list of definitions, one for each field, and each containing the following members: `value`, `latitude`, `longitude`. Where it is not possible to generate a sensible value due to lack of valid data in the fieldset, the grib missing indicator is returned.

The following example illustrates how to use the function.

```
fs = read (strGribFile)
listdef = nearest_gridpoint_info(fs, 51.46, -1.33)
loop ngp in listdef
  print ("Value : ", ngp.value)
  print ("Latitude : ", ngp.latitude)
  print ("Longitude : ", ngp.longitude)
end loop
```

fieldset *nobitmap* (fieldset,number)

Returns a copy of the input fieldset (first argument) with all of its missing values replaced with the number specified by the second argument. See also *bitmap*.

fieldset *pressure*(fieldset)
 fieldset *pressure*(fieldset,number)
 fieldset *pressure*(fieldset,list)
 fieldset *pressure*(fieldset,fieldset)

This function creates fields of pressure from the logarithm of the surface pressure (`lnsp`) and a list of model levels. *Note that this function only works with lat/long grids and assumes that the parameter for `lnsp` is 152. A newer, more flexible version of this function exists - see `unipressure` ().*

- The first argument is always a fieldset containing an `lnsp` field. If no other parameter is given, the list of levels will range from 1 to $(\text{number of vertical coordinates}/2)-1$ as coded in the GRIB header of the `lnsp` parameter.
- The second argument specifies the levels at which the output fields must be generated. To generate a single level, pass a number. For more than one level, either pass a list of levels or a fieldset. If a fieldset is passed as the second parameter, the level information is extracted from each field of the fieldset.

Missing values in the `lnsp` field are retained in the output fieldset.

```
fieldset putksec1( fieldset,list )
fieldset putksec2( fieldset,list )
fieldset putksec3( fieldset,list )
fieldset putksec4( fieldset,list )
```

Note: as of Metview 3.11, these functions are deprecated - please use the new alternatives, ***grib_set_long*** and ***grib_set_double*** on page II- 148.

This function allows you to set integer elements of the GRIB header Sections in the input fieldset to suitable values specified in the list provided as the second argument. Details on the GRIB section element numbers and their meaning can be found in :

```
http://www.ecmwf.int/publications/manuals/libraries/gribex/ksec1.html
http://www.ecmwf.int/publications/manuals/libraries/gribex/ksec2.html
http://www.ecmwf.int/publications/manuals/libraries/gribex/ksec3.html
http://www.ecmwf.int/publications/manuals/libraries/gribex/ksec4.html
```

The structure of the list provided as the second argument is as follows - the index of a GRIB Section element followed by its new value; these element pairs can be repeated as many times as necessary. Example :

```
f = putksec1(f,[10,98,18,113])
```

will assign 98 to `ksec1(10)` and 113 to `ksec1(18)`. This function does not modify the input fieldset, but returns a new fieldset with the modifications applied.

```
fieldset putrsec1( fieldset,list )
fieldset putrsec2( fieldset,list )
fieldset putrsec3( fieldset,list )
fieldset putrsec4( fieldset,list )
```

Note: as of Metview 3.11, these functions are deprecated - please use the new alternatives, ***grib_set_long*** and ***grib_set_double*** on page II- 148.

This function allows you to set real-valued elements of the GRIB header Sections in the input fieldset to suitable values specified in the list provided as the second argument. Details on the GRIB section element numbers and their meaning can be found in :

```
http://www.ecmwf.int/publications/manuals/libraries/gribex/psec2.html
http://www.ecmwf.int/publications/manuals/libraries/gribex/psec3.html
http://www.ecmwf.int/publications/manuals/libraries/gribex/psec4.html
```

The structure of the list provided as the second argument is as follows - the index of a GRIB Section element followed by its new value; these element pairs can be repeated as many times as necessary. Example :

```
f = putrsec2(f,[12,435,13,520])
```

will assign 435 to rsec2(12) and 520 to rsec2(13). This function does not modify the input fieldset, but returns a new fieldset with the modifications applied.

fieldset *rmask*(fieldset,number,number,number)
 fieldset *rmask*(fieldset,list)

Similar to *mask*, except that a round mask is computed with a given radius around a geographical centre point. These can be given by either:

- three numbers : latitude, longitude (in degrees), radius (in meters)
- a list containing the above three numbers

The name of this function is derived from the fact that it creates a “round mask” or a “radius mask”.

fieldset *rms*(fieldset)

Computes the root mean square of a fieldset. A missing value in any field will result in a missing value in the corresponding place in the output fieldset. With n fields in the input fieldset, if x_{ik} is the i th value of the k th input field and y_i is the i th value of the resulting field, the formula can be written :

$$y_i = \sqrt{\frac{1}{n} \sum_{k=1}^n (x_i^k)^2}$$

Note that the following lines are equivalent :

```
y = rms(x)
y = sqrt(mean(x*x))
```

fieldset *sinlat*(fieldset)

For each field in the input fieldset, this function creates a field where each grid point has the value of the sine of its latitude. For example, the following macro adds the coriolis parameter to each grid point of a field :

```
# Computes absolute vorticity from vorticity
omega      = 2 * pi / 86400
coriolis   = 2 * omega * sinlat(vort)
absvort    = vort + coriolis
```

fieldset *sort*(fieldset)
 fieldset *sort*(fieldset,string)
 fieldset *sort*(fieldset,list)
 fieldset *sort*(fieldset,string,string)
 fieldset *sort*(fieldset,list,string)
 fieldset *sort*(fieldset,list,list)

This function accepts a fieldset as input and returns it sorted according to keys and rules specified in the other input arguments.

Specified with only the fieldset as its single argument, `sort()` sorts in descending order the fieldset according to date, time, step, (ensemble) number, level and parameter.

The second argument allows you to modify the precedence of the sorting keys - e.g. if the second argument is a string "parameter", then the sorting is done according to the parameter first. If the second argument is a list you specify a list of sorting keys - e.g. ["parameter", "date"] sorts on parameter and then date.

The third argument specifies a sorting direction. This can be a string (">" or "<") or a list ([">", "<", ">", ...]). If it is a string, the sorting direction it specifies applies to all sorting keys specified in the second argument. If it is a list, then the second argument *must* also be a list with the same number of elements - the sorting directions apply to each sorting key specified.

fieldset *stdev*(fieldset)

Computes the standard deviation of a fieldset. A missing value in any field will result in a missing value in the corresponding place in the output fieldset. With n fields in the input fieldset, if x_{ik} is the i th value of the k th input field and y_i is the i th value of the resulting field, the formula can be written :

$$y_i = \sqrt{\frac{1}{n} \sum_{k=1}^n (x_i^k)^2 - \left(\frac{1}{n} \sum_{k=1}^n x_i^k \right)^2}$$

Note that the following lines are equivalent :

$$\begin{aligned} y &= \text{stdev}(x) \\ y &= \text{sqrt}(\text{mean}(x*x) - \text{mean}(x) * \text{mean}(x)) \\ y &= \text{sqrt}(\text{var}(x)) \end{aligned}$$

number or list *stdev_a*(fieldset) number or list *stdev_a*(fieldset,list)

Computes the standard deviation over a weighted area. The area, if specified, is a list of numbers representing North, West, South, East. If the area is not specified, the whole field will be used in the calculation. The result is a number for a single field, or a list for a multi-field fieldset.

fieldset *sum* (fieldset)

Computes the sum of a fieldset. The result is a single-field fieldset. A missing value in any field will result in a missing value in the corresponding place in the output fieldset. With n fields in the input fieldset, if x_{ik} is the i th value of the k th input field and y_i is the i th value of the resulting field, the formula can be written :

$$y_i = \sum_{k=1}^n x_i^k$$

fieldset *thickness*(fieldset)
 fieldset *thickness*(fieldset,number)
 fieldset *thickness*(fieldset,list)
 fieldset *thickness*(fieldset,fieldset)

This function creates fields of thickness from the logarithm of the surface pressure (`lnsp`) and a list of model levels. *Note that this function only works with lat/long grids and assumes that the parameter for `lnsp` is 152. A newer, more flexible version of this function exists - see `unithickness` ().*

- The first argument is always a fieldset containing an `lnsp` field. If no other parameter is given, the list of levels will range from 1 to $(\text{number of vertical coordinates}/2)-1$ as coded in the GRIB header of the `lnsp`.
- The second argument specifies the levels at which the output fields must be generated. To generate a single level, pass a number. For more than one level, either pass a list of levels or a fieldset. If a fieldset is passed as the second parameter, the level information is extracted from each field of the fieldset.

Missing values in the `lnsp` field are retained in the output fieldset.

fieldset *unipressure*(fieldset)
 fieldset *unipressure*(fieldset,fieldset)
 fieldset *unipressure*(fieldset,list)
 fieldset *unipressure*(fieldset,number)
 fieldset *unipressure*(fieldset,fieldset,number)
 fieldset *unipressure*(fieldset,list,number)

This function creates fields of pressure from the logarithm of the surface pressure (`lnsp`) and a list of model levels. Unlike `pressure` (), this function works with all grid types known to Metview (not just lat/long); it also allows the user to override the parameter number for `lnsp` (default 152).

- The first argument is always a fieldset containing an `lnsp` field. If no other parameter is given, then pressure is computed for all model levels that are described in the GRIB header of `fieldset`.
- If `number` is given (always the last parameter) it is the `lnsp` parameter code (default is 152).
- `list` should contain model levels for which pressure is to be computed. Note that also for a single model level one has to use a list (this is a signature difference compared to the old function `pressure` ()).
- If `fieldset` is given as the second parameter then pressure is computed for those model levels found in the second fieldset.

Missing values in the `lnsp` field are retained in the output fieldset.

fieldset *unithickness*(fieldset)
 fieldset *unithickness*(fieldset,fieldset)
 fieldset *unithickness*(fieldset,list)
 fieldset *unithickness*(fieldset,number)
 fieldset *unithickness*(fieldset,fieldset,number)
 fieldset *unithickness*(fieldset,list,number)

This function creates fields of thickness from the logarithm of the surface pressure (`lnsp`) and a list of model levels. Unlike `thickness` (), this function works with all grid types known to

Metview (not just lat/long); it also allows the user to override the parameter number for `lnsp` (default 152).

- The first argument is always a fieldset containing an `lnsp` field. If no other parameter is given, then thickness is computed for all model levels that are described in the GRIB header of `fieldset`.
- If `number` is given (always the last parameter) it is the `lnsp` parameter code (default is 152).
- `list` should contain model levels for which thickness is to be computed. Note that also for a single model level one has to use a list (this is a signature difference compared to the old function `thickness()`).
- If `fieldset` is given as the second parameter then thickness is computed for those model levels found in the second fieldset.

Missing values in the `lnsp` field are retained in the output fieldset.

fieldset *univertint*(fieldset)
 fieldset *univertint*(fieldset, fieldset)
 fieldset *univertint*(fieldset, number)
 fieldset *univertint*(fieldset, fieldset, number)
 fieldset *univertint*(fieldset, fieldset, list)

Universal vertical integration. This function is similar to `vertint()`, except that `univertint()` also works with sparse levels whereas `vertint()` is restricted to continuous levels.

This function performs a vertical integration of the input fieldset. If just one fieldset is given, it must also contain the logarithm of the surface pressure (`lnsp`). If the function is called with two fieldsets, the first one is a fieldset containing an `lnsp` field, the second one is the multi-level fieldset. A number may optionally be given as the last parameter in order to specify the `lnsp` code used in the fieldset that contains the `lnsp` data; its default value is 152. If a list with two elements [`top`, `bottom`] is given as the third parameter, then the integration is performed between (and including) these layers.

A missing value in any field will result in a missing value in the corresponding place in the output fieldset.

The function computes :

$$\int_{bot}^{top} (f) \frac{dp}{g}$$

Where f is the fieldset, p is the pressure and g the acceleration of gravity.

fieldset *var*(fieldset)

Computes the variance of a fieldset. A missing value in any field will result in a missing value in the corresponding place in the output fieldset. With n fields in the input fieldset, if x_{ik} is the i th value of the k th input field and y_i is the i th value of the resulting field, the formula can be written :

$$y_i = \frac{1}{n} \sum_{k=1}^n (x_i^k)^2 - \frac{1}{n} \left(\sum_{k=1}^n x_i^k \right)^2$$

Note that the following lines are equivalent :

$$\begin{aligned} y &= \text{var}(x) \\ y &= \text{mean}(x*x) - \text{mean}(x) * \text{mean}(x) \end{aligned}$$

number or list **var_a**(fieldset)
number or list **var_a**(fieldset,list)

Computes the variance over a weighted area. The area, if specified, is a list of numbers representing North, West, South, East. If the area is not specified, the whole field will be used in the calculation. The result is a number for a single field, or a list for a multi-field fieldset.

fieldset **vertint**(fieldset)
fieldset **vertint**(fieldset,fieldset)

This function performs a vertical integration of the input fieldset, which must contain a range of model levels for the same parameter. A missing value in any field will result in a missing value in the corresponding place in the output fieldset. If the function is called with the fieldset as its single argument, it must also contain the logarithm of the surface pressure (`lnsp`). If the function is called with two parameters, the first one is a fieldset containing an `lnsp` field, the second one is the multi-level fieldset.

The function computes :

$$\int_{bot}^{top} (f) \frac{dp}{g}$$

Where f is the fieldset, p is the pressure and g the acceleration of gravity.

The following example computes the total amount of liquid water in the atmosphere by integrating the cloud liquid water content (`clwc`) over all levels of the model :

```
# Retrieve clwc
clwc = retrieve(
    levtype      : "ml",
    levelist     : [1,"to",31],
    param        : "clwc",
    date         : -1,
    grid         : [2.5,2.5]
)

# Retrieve lnsp
```

```
lnsp = retrieve(  
    levtype   : "ml",  
    levelist  : 1,  
    param     : "lnsp",  
    date      : -1,  
    grid      : [2.5,2.5]  
)  
  
# Integrate the field  
x = vertint(lnsp,clwc)  
plot(x)
```


Functions and Operations on Images

Note

Because each pixel has a value between 0 and 1 (see "Observations" on page II- 22), and because any result is clipped to this interval, some functions such as `abs()`, `neg()`, `int()` and `sgn()`, although valid, are meaningless. Other functions, although available, may not lead to sensible outcomes.

`image (image op image)`

Operation between two images. *op* is one of the operators below :

+	Addition	-	Subtraction
*	Multiplication	/	Division
^	Power		

The images returned by these boolean operators are boolean images (containing only 1 where result is true, 0 where it is false) :

>	Larger Than	<	Smaller Than
>=	Larger or Equal	<=	Smaller or Equal
=	Equal	<>	Not Equal

`image (image op number)`

`image (number op image)`

Operations between images and numbers. *op* is any of the operations defined above.

`geopoints (image op geopoints)`

`geopoints (geopoints op image)`

Operations between images and geopoints. *op* is any of the operations defined above.

`image (image and image)`

`image (image or image)`

`image (not image)`

Conjunction, Disjunction and Negation. Boolean operators consider all null values to be false and all non null values to be true. The images created by boolean operators are binary images (containing only 1 where result is true, 0 where it is false).

`image (image and number)`

`image (number or image)`

Conjunction and Disjunction. Boolean operations between images and numbers. See above.

geopoints (image **and** geopoints)
geopoints (geopoints **or** image)

Boolean operations between images and geopoints.

image **asin**(image)
image **acos**(image)
image **atan**(image)

Return the arc trigonometric function of the input image at each grid point. Result is in radians.

image **cos**(image)

Return the image of the cosine of the input image. Input image must have values in radians.

image **exp**(image)

Returns the image of the exponential of the input image.

image **log**(image)

Returns the image of the natural log of the input image.

image **log10**(image)

Returns the image of the base 10 log of the input image.

image **sin**(image)

Return the image of the sine of the input image. Input image must have values in radians.

image **sqrt**(image)

Returns the image of the square root of the input image.

image **tan**(image)

Return the image of the tangent of the input image. Input image must have values in radians.

image **convolution**(image,list)

Applies a convolution matrix to an image. The second parameter must be a list of 9 numbers, [n11, n12, n13, n21, n22, n23, n31, n32, n33] representing the matrix :

$$\begin{vmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{vmatrix}$$

For each pixel p_{ij} , the resulting pixel q_{ij} is computed as:

$$q_{ij} = \sum_{k=1}^3 \sum_{l=1}^3 n_{kl} \cdot P_{(i+k-2)(j+l-2)}$$

For example, you can apply the Laplacian operator:

```
l = convolution( image, [-1,-1,-1,-1,8,-1,-1,-1,-1] )
```

to highlight the contours of the image features.

image filter(image,list)

Remaps an image to the values specified in the list entered as the second argument. This list must contain 256 values. A new image is produced with each pixel equal to the n th value of the input list, where n is the input image pixel value.

image max(image,image)

image min(image,image)

Returns the image of the maximum (minimum) of two images.

image max(image,number)

image min(image,number)

Returns the image of the maximum (minimum) of number and the pixel values.

image reduce(image,number)

This function reduces the size of an image by a given scaling factor. The following code:

```
half = reduce(image,2)
```

produces an image with half the number of pixels in each dimension (thus four times smaller in memory and disk size). This function does not perform processing on the data. If the second parameter is n , it extracts every n th pixel from the source image.

Functions and Operators on Observations

Due to the nature of the data, the macro language provides little in the way of functions to process observations - you can only merge and/or plot them.

If you need to do more, you have to convert them to geopoints using the `obsfilter()` request.

observations (observations & observations)
observations *merge*(observations,observations)

Merges two sets of observations.

observations *obsfilter*(definition)
geopoints *obsfilter*(definition)

The `obsfilter()` function is a Metview request, i.e. corresponds to a Metview icon available from the User Interface - Observation Filter. It accepts `BUFR` input, filters out part of that input (optional) and returns `BUFR` data (default) or geopoints (if user so specifies) in one of three geopoints formats - see "Geopoints" on page II- 23.

The input is a definition containing a named list of values, corresponding to the contents of the Observation Filter editor contents. Remember you need only specify the parameters which you want to change from their default values.

Functions and Operators on Geopoints

See first "Geopoints" on page II- 23 for more details.

geopoints (geopoints *op* geopoints)

Operation between two geopoints. *op* is one of the following :

+	Addition	-	Subtraction
*	Multiplication	/	Division
^	Power		

The geopoints returned by these boolean operators are boolean geopoints (containing only 1 where result is true, 0 where it is false) :

>	Larger Than	<	Smaller Than
>=	Larger or Equal	<=	Smaller or Equal
=	Equal	<>	Not Equal

geopoints (geopoints *op* number)
geopoints (number *op* geopoints)

Operations between geopoints and numbers. *op* is any of the operations defined above. Missing values retain their value of `geo_missing_value`.

geopoints (geopoints *op* fieldsets)
geopoints (fieldsets *op* geopoints)

Operations between geopoints and fieldsets. *op* is any of the operations defined above. Missing values, both in the fieldset and in the original geopoints variable result in a value of `geo_missing_value`.

geopoints (geopoints *and* geopoints)
geopoints (geopoints *or* geopoints)
geopoints (*not* geopoints)

Conjunction, Disjunction and Negation. Boolean operations on geopoints consider all null values to be false and all non null values to be true. Missing values retain their value of `geo_missing_value`.

geopoints (geopoints & geopoints & ...)
geopoints (nil & geopoints & ...)
geopoints (geopoints & nil)
geopoints *merge*(geopoints,geopoints,...)

Merge several sets of geopoints. The output is the concatenation of each set of geopoints. Merging with the value `nil` does nothing, and can be used to initialise when building a set of geopoints in

a loop. Note that only geopoints that are in the same format can be merged. See "Geopoints" on page II- 23 for details of the different formats.

definition `geopoints[number]`

Returns a definition with values of the `n`th point of the geopoints. *Note that, unlike lists, the first geopoint is at index 0.*

`geopoints abs(geopoints)`

Returns the geopoints of the absolute value of the input geopoints. Missing values retain their value of `geo_missing_value`.

`geopoints asin(geopoints)`

`geopoints acos(geopoints)`

`geopoints atan(geopoints)`

Returns the geopoints of the arc trigonometric function of the input geopoints. Result is in radians. Missing values retain their value of `geo_missing_value`.

`geopoints cos(geopoints)`

Return the cosine of the input geopoints. These must be in radians. Missing values retain their value of `geo_missing_value`.

`geopoints exp(geopoints)`

Returns the geopoints of the exponential of the input geopoints. Missing values retain their value of `geo_missing_value`.

`geopoints int(geopoints)`

Returns the geopoints of the integer part of the input geopoints. Missing values retain their value of `geo_missing_value`.

`number intbits(geopoints,number)`

`number intbits(geopoints,number,number)`

Takes the integer part of the geopoints values and extracts a specified bit (or number of bits if a second number parameter is specified), where bit number 1 is the least significant bit (lsb). A single bit will always be returned as 1 or 0, regardless of its position in the integer. A group of bits will be treated as if the first bit is the least significant bit of the result. A few examples from the 'number' version of this function illustrate.

To extract the 1st, 2nd and 3rd bits from a number separately:

```
n = 6 # in bit-form, this is '00000110' with the lsb at the right
flag = intbits (n, 1) # flag is now 0
flag = intbits (n, 2) # flag is now 1
```

```
flag = intbits (n, 3) # flag is now 1
```

To extract the 1st and 2nd bits together to make a single number:

```
flag = intbits (n, 1, 2) # flag is now 2
```

To extract the 2nd and 3rd bits together to make a single number:

```
flag = intbits (n, 2, 2) # flag is now 3
```

To extract the 3rd and 4th bits together to make a single number:

```
flag = intbits (n, 3, 2) # flag is now 1
```

The number of bits available depends on the machine architecture and Metview's compilation options, but at the time of writing it should be 32. This function does not treat missing values differently from any other values (for efficiency with large datasets).

geopoints **log**(geopoints)

Returns the geopoints of the natural log of the input geopoints. Missing values retain their value of `geo_missing_value`.

geopoints **log10**(geopoints)

Returns the geopoints of the base 10 log of the input geopoints. Missing values retain their value of `geo_missing_value`.

geopoints **neg**(geopoints)

Returns the geopoints of the negative of the input geopoints. The same as `(-geopoints)`. Missing values retain their value of `geo_missing_value`.

geopoints **sgn**(geopoints)

Returns the geopoints of the sign of the values of the input geopoints : -1 for negative values, 1 for positive and 0 for null values. Missing values retain their value of `geo_missing_value`.

geopoints **sin**(geopoints)

Return the sine of the input geopoints. These must be in radians. Missing values retain their value of `geo_missing_value`.

geopoints **sqrt**(geopoints)

Returns the geopoints of the square root of the input geopoints. Missing values retain their value of `geo_missing_value`.

geopoints *tan*(geopoints)

Return the tangent of the input geopoints. These must be in radians. Missing values retain their value of `geo_missing_value`.

number *count*(geopoints)

Returns the total number of elements in the geopoints.

**geopoints *create_geo*(number)
geopoints *create_geo*(number, string)**

Creates a new geopoints variable with the given number of points, all set to default values and coordinates. It is intended that this function be used in conjunction with the `set_XXX` geopoints functions in order to populate the geopoints with data. If saved, the geopoints file will be in the 'traditional' 6-column format. If another format is desired, supply a string as the second parameter, possible values being 'polar_vector', 'xy_vector' and 'xyv'.

list *date*(geopoints)

Extracts the date information of all the geopoints and returns it as a list of dates.

**string or list *db_info*(geopoints,string)
string *db_info*(geopoints,string,string)**

Returns information about the database retrieval which generated the geopoints. The first string parameter specifies which piece of information you would like; possible values are:

Table I-1 :

name	the name of the database system, e.g. ODB
path	the path to the database
query	a list of strings containing the multi-line data query
column	the name of the database column used to populate a given element of the geopoints. A second string must be provided, naming the geopoints element of interest - possible values are 'lat', 'lon', 'level', 'date', 'time', 'value' and 'value2'.
alias	similar to column above, but returns the name of the database alias used instead of the full column name

Note that this information is derived from the DB_INFO section (if it exists) in the geopoints file header (see "Storing Data Origin Information in a Geopoints File" on page II- 25).

**geopoints *distance*(geopoints,number,number)
geopoints *distance*(geopoints,list)**

Returns geopoints with the value of each point being the distance in meters from the given geographical location. The location may be specified by supplying either two numbers (latitude and longitude respectively) or a 2-element list containing latitude and longitude in that order. The location should be specified in degrees.

geopoints *filter*(geopoints,geopoints)

A filter function to extract a subset of its geopoints input using a second geopoints as criteria. The two input geopoints must have the same number of values. The resulting output geopoints contains the values of the first geopoints where the value of the second geopoints is non-zero. It is usefully employed in conjunction with the comparison operators :

```
freeze = filter(temperature,temperature < 273.15)
```

The variable `freeze` will contain a subset of `temperature` where the value is below 273.15. The following example shows how to plot a geopoints set with different colours :

```
# Filter from "temperature" points at, above, below 273.15
cold = filter( temperature,temperature<273.15 )
zero = filter( temperature,temperature=273.15 )
warm = filter( temperature,temperature>273.15 )

# Create three symbol plotting definitions
red = psymb( symbol_colour : "red" )
blue = psymb( symbol_colour : "blue" )
black = psymb( symbol_colour : "black" )

# Plot everything
plot( zero,black,cold,blue,warm,red)
```

geopoints *filter*(geopoints,number)**geopoints** *filter*(geopoints,list)

A filter function to extract a subset of its geopoints input using model levels as criteria.

- If the second argument is a number, the function extracts all the geopoints for which the level is equal to the number.
- If the second argument is a list of two numbers [n1,n2], the function extracts all the geopoints for which the level lies in the n1-n2 interval.

geopoints *filter*(geopoints,date)**geopoints** *filter*(geopoints,list)

A filter function to extract a subset of its geopoints input using dates as criteria.

- If the second argument is a date, the function extracts all the geopoints for which the date is equal to the one specified as the second argument.
- If the second argument is a list of two dates [d1,d2], the function extracts all the geopoints for which the date lies in the d1-d2 interval.

geopoints *filter*(geopoints,list)

A filter function to extract a subset of its geopoints input using a geographical area as criteria.

The second argument is a list of four numbers (lat/long coordinates) defining a geographical area - [North, West, South, East]. The function extracts all the geopoints that fall within the specified area.

geopoints *find*(fieldset,number)
geopoints *find*(fieldset,number,number)

A filtering function that returns a geopoints holding the grid points whose value is equal to the *value* of the first number. If a second number is given as the third argument it is a tolerance *threshold* and the geopoints will hold the grid points for which :

```
abs(data-value) <= threshold
```

geopoints *geosort*(geopoints)

Returns a new geopoints variable that contains the input geopoints sorted geographically from North to South (and West to East in points with the same latitude value, then by height, with lowest numerical values first).

geopoints *interpolate*(fieldset,geopoints)

Generates a set of geopoints from a field. The first parameter must contain a single field. The field is interpolated for each position of the geopoints given as a second parameter. Where it is not possible to generate a sensible value due to lack of valid data in the fieldset, the internal geopoints missing value is used (this value can be checked for with the built-in variable `geo_missing_value` or removed with the function `remove_missing_values`).

list *latitude*(geopoints)

Extract the latitude of all the geopoints and returns it as a list of numbers.

list *level*(geopoints)

Extract the height of all the geopoints and returns it as a list of numbers.

list *longitude*(geopoints)

Extract the longitude of all the geopoints and returns it as a list of numbers.

geopoints *max*(geopoints,geopoints)
geopoints *min*(geopoints,geopoints)

Returns the geopoints of maximum (minimum) value at each point. Missing values retain their value of `geo_missing_value`.

geopoints *max*(geopoints,number)
geopoints *min*(geopoints,number)

Returns the geopoints of the maximum (minimum) of number and the geopoints value at each point. Missing values retain their value of `geo_missing_value`.

geopoints *max*(geopoints,fieldsets)
geopoints *min*(geopoints,fieldsets)

Returns geopoints of maximum (minimum) of the geopoints value and the geopoints value at each grid point or spectral coefficient. Missing values, either in the fieldset or in the original geopoints variable, result in a value of `geo_missing_value`.

number *maxvalue*(geopoints)
number *minvalue*(geopoints)

Returns the maximum (minimum) value of all geopoints values. Missing values are bypassed in this calculation. If there are no valid values, then `nil` is returned.

number *mean*(geopoints)

Computes the mean of the geopoints. Missing values are bypassed in this calculation. If there are no valid values, then `nil` is returned.

geopoints *offset*(geopoints,number,number)
geopoints *offset*(geopoints,list)

Modifies the locations of a set of geopoints by specified amounts. The offsets can be specified either as two separate numbers or as a 2-element list. The original geopoints variable is unaffected; the functions return a new variable.

geopoints *polar_vector*(geopoints, geopoints)

Combines two single-parameter geopoints variables into a polar vector style geopoints variable. The first represents speed, the second represents direction. Both input geopoints variables should contain the same number of points.

geopoints *remove_duplicates*(geopoints)

Returns a new geopoints variable that contains just one instance of any duplicate geopoint. Two geopoints are considered to be duplicates of each other if the files have the same format and the points have the same coordinates, height, date, time and values.

geopoints *remove_missing_values*(geopoints)

Returns a new geopoints variable that contains just the non-missing values from the input geopoints variable. A geopoint is considered to be missing if either its `value` or `value2` members are missing.

geopoints *set_latitude*(geopoints, number or list)
geopoints *set_longitude*(geopoints, number or list)
geopoints *set_level*(geopoints, number or list)
geopoints *set_date*(geopoints, number or list)
geopoints *set_time*(geopoints, number or list)
geopoints *set_value*(geopoints, number or list)
geopoints *set_value2*(geopoints, number or list)

Returns a geopoints variable with either its latitude, longitude, level, date, time, value or value2 component modified.

All these functions take two parameters: first one must be a geopoints variable, the second can be a number or a list of numbers. If a number is given then all the corresponding values (latitude, longitude, level, or ...) are replaced by the given value.

If a list is given as the second parameter then the corresponding values are replaced from the given list. If the list is shorter than geopoints then only the first values that have a corresponding value in the list, are changed.

NOTE: for dates, 8 digit integers must be used. If the list contains non-numbers, then a missing value is written into the corresponding geopoints value.

geopoints *subsample*(geopoints, geopoints)

Returns a geopoints variable containing the same locations (latitude, longitude and height) as the second geopoints variable, but whose values are from the first geopoints variable (or a missing value if point not found in the first variable). Note that the resulting geopoints variable is sorted in the same way as performed by the *geosort*() function. This means that you need to be careful if performing functions between the results of a *subsample*() operation and another geopoints variable; if the locations in the two geopoints are the same, then you should *geosort*() the second geopoints beforehand.

You can use function *remove_missing_values*() if you need to get rid of the missing valued points in the returned geopoints variable.

number *sum*(geopoints)

Computes the sum of the geopoints. Missing values are bypassed in this calculation. If there are no valid values, then *nil* is returned.

list *value*(geopoints)

Extracts the value of all the geopoints and returns it as a list of numbers.

list *value2*(geopoints)

Extracts the second value of all the geopoints and returns it as a list of numbers.

geopoints *xy_vector*(geopoints, geopoints)

Combines two single-parameter geopoints variables into a u/v style geopoints variable. Both input geopoints variables should contain the same number of points.

Functions and Operators on NetCDF

See first "How operators and functions work on NetCDF" on page II- 27.

`netcdf (netcdf op netcdf)`

Operation between two netcdf data units. *op* is one of the following :

+	Addition	-	Subtraction
*	Multiplication	/	Division
^	Power		

The netcdf returned by these boolean operators are boolean netcdf (containing only 1 where result is true, 0 where it is false) :

>	Larger Than	<	Smaller Than
>=	Larger or Equal	<=	Smaller or Equal
=	Equal	<>	Not Equal

`netcdf (netcdf op number)`

`netcdf (number op netcdf)`

Operations between netcdf and numbers. *op* is any of the operations defined above

`netcdf (netcdf and netcdf)`

`netcdf (netcdf or netcdf)`

Conjunction and Disjunction. Boolean operators consider all null values to be false and all non null values to be true. The netcdf created by boolean operators are binary netcdf (containing only 1 where result is true, 0 where it is false). For example :

```
netcdf3 = netcdf1 and netcdf2
```

creates a netcdf `netcdf3` with values of 1 where the corresponding values of fieldset `netcdf2` and `netcdf1` are both non zero, and 0 otherwise.

definition `attributes(netcdf)`

This function returns a definition variable holding the metadata of the input netcdf, e.g. date, time, levels, etc, .

```
attr_list = attributes(netcdf)
date = attr_list.DATE
levelist = attr_list.LEVELIST
```

list *dimensions*(netcdf)

This returns a list of numbers, each representing one of the dimensions of the data held in the netcdf :

```
dim_list = dimensions(netcdf)
```

e.g. if the current variable is a cross section, the output list would have two numbers, the first the number of levels, the second the number of points along the horizontal.

netcdf *max*(netcdf,netcdf)**netcdf *min*(netcdf,netcdf)**

Returns the netcdf of maximum (minimum) value of the two input netcdf.

netcdf *max*(netcdf,number)**netcdf *min*(netcdf,number)**

Returns the netcdf of the maximum (minimum) of the number and the netcdf values.

netcdf *mod*(netcdf,netcdf)

Returns a netcdf whose values are the remainder of the division of the first netcdf by the second netcdf .Where the values of the second netcdf are larger than those of the first, the output value is set to the integer part of the value of the first netcdf. Note that only the integer parts of the inputs are considered in the calculation, meaning that a second parameter of 0.5 would cause a division by zero.

none *setcurrent*(netcdf, number)

On multi-variable netcdfs this sets the variable specified by the number given as second argument as the current variable. *Functions and operators act on the current variable only.*

The netcdf produced by the Metview applications are uni-variable, so in their case this function need not be used. For multi-variable netcdf variables, `setcurrent()` can be usefully combined with the function `variables()` :

```
var_list = variables(netcdf)
for i = 1 to count(var_list) do
  setcurrent(netcdf, i)
  netcdf = netcdf - 273.15# acts on current variable only
end for
```

list *value* (netcdf, number)

Returns the nth (second parameter) value of the current netcdf variable.

list *values*(netcdf)

This returns a list with all the values held in the current variable :

```
val_list = values(netcdf)
```

You will need to use the output of the `dimensions()` function to be able to navigate within this output list. E.g. for the case of a two dimensional variable such as a cross-section, the following code loops over all its data values :

```
dim = dimensions(xsect)
xs_vals = values(xsect)
for j = 1 to dim[1] do
  for i = 1 to dim[2] do
    index = (j-1)*dim[2] + i
    val = xs_vals[index]
  end for
end for
```

list *variables*(netcdf)

Extracts the variable names of the variables contained in a netcdf and returns them as a list of strings. Count the number of elements in the output list to give you the number of variables. The netcdf produced by the Metview applications are uni-variable, so in their case this returns a single element list.

```
var_list = variables(netcdf)
print ("netcdf contains ", count(var_list), " variables")
```


Functions and Operators on Vectors and Matrices

matrix *matrix*(number,number)

Allocates a new matrix. The input arguments represent the dimensions of the matrix. The matrix is initialised with null values.

number *matrix*[number,number]

Returns the (m,n)th element of a matrix.

vector *vector*(number)

Allocates a new vector. The input argument is the dimension of the vector. The vector is initialised with null values.

number *vector*[number]

Returns the nth element of a vector.

number *count*(vector)

Returns the number of elements in a vector.

matrix (matrix * matrix)

Returns a matrix equal to the product of two matrices.

number *det*(matrix)

Returns a number equal to the determinant of a matrix

matrix *inverse*(matrix)

Returns a matrix equal to the inverse of a matrix

matrix *transpose*(matrix)

Returns a matrix equal to the transpose of a matrix

Functions on Definitions

A definition is a list of named items (which may be numbers, strings, lists,...). See "Definitions" on page II- 17 for examples and further details.

definition definition(any,...)

Builds a definition with the specified elements. Note that the keyword `definition` is not in fact required:

```
my_field = (param : "z", level : 500, grid : [2, 2])
```

any definition[string]

Returns the value named by the input string from a definition.

```
level = my_field["level"]
```

Note the shorthand form of this, the dot operator:

```
level = my_field.level
```


File I/O Functions

For a detailed explanation and examples please see "Arrays - Vectors and Matrices" on page II- 32.

none **append**(string,any,...)
 none **append**(filehandler,any,...)

Writes output to a file specified by its name or by a filehandler previously assigned to it by the `file()` function. The output file type depends on the type that is being written - in exactly the same way as it does for the `write()` function (see below). As the name implies, `append()` never overwrites previously existing output.

For further details see "Output: `write()` and `append()` Functions" on page II- 37.

number **exist**(string)

This function checks whether a file or directory exists. The single argument is the file or directory name - *you must specify the full path*. The function returns a number, 1 if the file exists and 0 otherwise. Use it combined with `fail()` or `stop()` for error checking :

```
if (not(exist("/home/xy/xyz/metview/grib_file"))) then
  fail("specified input file does not exist!")
end if
(...)
```

filehandler **file**(string)

Assigns a file handler to a file whose name is the function argument. The file handler can be used in place of the file name in file output functions - `write()` and `append()`.

none **newpage**(display window)

Forces a new page to be taken in the current PostScript file.

none **print**(...)

Prints all its arguments to the output area of the main user interface (and to that of any opened macro editor window).

fieldset	read (string)
observations	read (string)
geopoints	read (string)
list	read (string)
netcdf	read (string)

Reads a data file whose name is passed as the argument. If the file is in the same folder as the macro program the path needn't be specified. The function returns a variable of the corresponding type. You needn't specify anything about the data type, it is automatically detected by the function.

The variable of type list is used to hold the contents of an ASCII file - the elements of this list variable are themselves lists, each holding a line of text. The elements of these sub lists are the text line tokens (component strings) arising from the parsing of the text.

For further details see "Input: `read()` Function" on page II- 32.

string *tmpfile()*

Reserves and returns a unique file name (inside the Metview cache directory) for a temporary file. Returned filenames are unique even when there are several copies of the same macro being executed simultaneously.

none *write()* (string,any,...)

none *write()* (filehandler,any,...)

Writes output to a file specified by its name or by a filehandler previously assigned to it by the `file()` function. The output file type depends on the type that is being written - if it is a fieldset then it creates a GRIB file, if it is observations it creates a BUFR file, if geopoints creates a geopoints file, if it is anything else it will create a text file with the current value of the variable(s) - an icon (associated with the corresponding file type) is also created if the files are saved to the Metview directory structure.

If you use `write()` sequentially, note that it will overwrite any previous output if called with a file name, but will add to previous output if called with a filehandler.

For further details see "Output: `write()` and `append()` Functions" on page II- 37.

Timing Functions

For a detailed explanation of Macro's timing functionality, see "Performance and Timing" on page II- 117.

none *stopwatch_start*(string)

Starts and names the macro stopwatch. Prints the current date and time. Only one stopwatch can be used at a time - the name is used only for the purpose of printing meaningful information. Starting a new stopwatch stops an existing stopwatch.

none *stopwatch_laptime*(string)

Prints the laptime since the previous call to *stopwatch_laptime*(), or from *stopwatch_start*() if there is no previous laptime. The string argument is used in the printout to identify the laptime.

none *stopwatch_stop*()

Stops the stopwatch and prints the total times since *stopwatch_start*(), or from *stopwatch_reset*() if that has been called. Also, prints the current date and time.

none *stopwatch_reset*(string1)

Stops the stopwatch and restarts it with a new name. This is equivalent of calling *stopwatch_stop*() and then *stopwatch_start*() with a new name.

Metview Icon-Functions

Metview icon-functions are functions which correspond to Metview icons like Mars Retrieval or Contour. The arguments of a Metview icon-function are specified like the elements of a definition variable (see "Definitions" on page II- 17) :

```
w = retrieve(date : -1, param : ["u","v"])
```

Or in an equivalent way, an icon-function accepts a definition as its argument :

```
wind = (date : -1, param :["u","v"])# this is a definition variable
w = retrieve(wind)
```

See "Icon-Functions" on page II- 50 for examples and further details.

Auxiliary functions

string *class*(definition)

Returns the class of a definition variable, i.e. such as returned by an icon-function.

```
a = icon(
      name      : "contour",
      exclusive : "false"
    )
input = dialog([a])

visdef = input["contour"]
if class(visdef) <> "PCONT" then
  fail ("icon must be a contour")
end if
```

list *values*(string)

Returns a list of all component parameters of a Metview request. The input argument string is the name of the request, e.g. : `values("retrieve")`. See "Information on icon-function input" on page II- 52.

list *values*(string,string)

Returns the list of all the possible values that a given parameter of a Metview request may take. The input argument strings are the name of the request and the name of the parameter, e.g. :

```
values( "retrieve","param" ).
```

The two functions above allow you to investigate the possible specification of an icon-function or of its parameters. See "Information on icon-function input" on page II- 52.

Next we provide a list of the currently available icon-functions.

Data access and data filtering icon-functions

fieldset *retrieve*(...)

Retrieves some specified data from ECMWF archives via MARS. Corresponds to the Metview icon MARS Retrieval.

fieldset *read*(...)

Reads in model output data from the ECMWF archive or from a GRIB file, with the option of filtering its contents. Returns fieldsets. Corresponds to the Metview icon GRIB Filter.

ecfs(...)

Retrieves a file from the ECFS - ECMWF's user file archive. Corresponds to the Metview icon Ecfs.

observations/geopoints *obsfilter*(...)

Reads in observation data from the ECMWF archive or from a BUFR file, with the option of filtering its contents. Returns either BUFR (observations) or geopoints. Corresponds to the Metview icon Observation Filter.

geopoints *odb*(...)

Retrieves data from an ODB database. Returns geopoints. Corresponds to the Metview icon ODB Access.

fieldset *geo_to_grib*(...)

Converts a geopoints variable to a fieldset by interpolation to a regular grid at a user specified resolution. Corresponds to the Metview icon Geopoints to GRIB.

geopoints *grib_to_geo*(...)

Converts GRIB data into geopoints format. Corresponds to the Metview icon GRIB to Geopoints.

Data processing icon-functions

netcdf *cross_sect*(...)

Generates a vertical cross-section data unit (NetCDF format) from a fieldset of upper air fields. The cross-section is defined over a specified pressure range for a transect line. Corresponds to the Metview icon Cross Section.

netcdf *xs_average*(...)

Generates an average vertical cross-section data unit from a fieldset of upper air fields. The average is taken over the N-S or the E-W directions for a specified rectangular area. The cross-section is defined over a specified pressure range for a transect line. Corresponds to the Metview icon Average.

netcdf *hovmoeller_line*(...)

Generates a Hovmøller diagram data unit along a specified arbitrary transect line. Corresponds to one of the options of the Metview family icon Hovmøller Data.

netcdf *hovmoeller_area*(...)

Generates a Hovmøller diagram data unit over a specified rectangular area. Corresponds to one of the options of the Metview family icon Hovmøller Data.

netcdf *hovmoeller_height*(...)

Generates a Hovmøller diagram data unit over a specified rectangular area, given a fieldset of one parameter on several levels (pressure or model levels). The value of each point in the diagram corresponds to the mean over the input area related to one level and one time. Corresponds to one of the options of the Metview family icon Hovmøller Data.

netcdf *hovmoeller_expand*(...)

Updates a Hovmøller diagram data unit previously generated by one of the options of the Metview family icon Hovmøller Data.

fieldset *percentile*(...)

Returns a fieldset of percentiles from an input fieldset. Corresponds to the Metview icon Percentile.

netcdf *vert_prof*(...)

Generates a vertical profile data unit (NetCDF format) from a fieldset of upper air fields. The profile is defined over a specified pressure range for a geographical point or area. Corresponds to the Metview icon Vertical Profile.

fieldset *pott_m*(...)

Returns a fieldset of potential temperature from $\ln(\text{surface pressure})$ and temperature data. Corresponds to one of the options of the Metview family icon Potential Temperature.

fieldset *eqpott_m*(...)

Returns a fieldset of equivalent potential temperature from $\ln p$, temperature and humidity data. Corresponds to one of the options of the Metview family icon Potential Temperature.

fieldset *seqpott_m*(...)

Returns a fieldset of saturated equivalent potential temperature from $\ln(\text{surface pressure})$ and temperature data. Corresponds to one of the options of the Metview family icon Potential Temperature.

fieldset *relhum*(...)

Returns a fieldset of relative humidity from its input data of temperature, specific humidity and $\ln p$ (only required if the input data is specified in model levels). The relative humidity is defined as in the IFS model output, i. e. with respect to water above 273.16 K, to ice below 250.16 K, and

with respect to a mixture of both in between. Corresponds to one of the options of the Metview family icon Relative Humidity.

fieldset *divrot*(...)

Returns rotational wind vectors from vorticity data. Corresponds to one of the two options of the Metview family icon Rotational Wind / Divergent Wind.

fieldset *divwind*(...)

Returns divergent wind vectors from divergence data. Corresponds to one of the two options of the Metview family icon Rotational Wind / Divergent Wind

fieldset *velpot*(...)

Returns a fieldset of velocity potential from divergence data. Corresponds to one of the options of the Metview family icon Velocity Potential / StreamFunction

fieldset *streamfn*(...)

Returns a fieldset of streamfunction from vorticity data. Corresponds to one of the options of the Metview family icon Velocity Potential / StreamFunction

list *stations*(...)

Returns a list of geographical locations (whether they correspond or not to actual meteorological stations). Corresponds to the Metview icon Stations.

Data plotting icon-functions

list *metgram*(...)

Generates a plot of the time series of specified meteorological variables. Corresponds to the Metview icon Metgram. As of Metview 3.7.2, this functionality has been superseded by the following two functions.

list *fieldpoint_timeseries*(...)

Generates a plot of the time series of specified meteorological variables stored as fieldsets. Corresponds to the Metview icon Time Series.

list *geopoint_timeseries*(...)

Generates a plot of the time series of specified meteorological variables stored as geoints. Corresponds to the Metview icon Time Series.

metgram0(...)

This macro function has been deprecated. See the *meteogram*() function instead.

meteogram(...)

Generates an epsgram, metgram or wavegram chart - these are standard plots containing time series (box-plots in the case of EPS data) of various meteorological parameters. Corresponds to the Meteogram icon.

epsmetgram(...)

This macro function has been deprecated. See the ***meteogram()*** function instead.

eps_metgram_coach(...)

This macro function has been deprecated. See the ***meteogram()*** function instead.

definition *vector_field(...)*

Generates a vectorial field from a pair of U and V components, using wind arrows as the visual definition. Corresponds to one of the options of the Metview family icon Vector.

definition *polar_field(...)*

Generates a vector field from a pair of intensity and direction fields, using arrows as the visual definition. Corresponds to one of the options of the Metview family icon Vector.

list *colour_vector_field(...)*

Generates a vectorial field from a pair of U and V components, using arrows as the visual definition, but coloured according to the value of a third scalar field (e.g. temperature). Corresponds to one of the options of the Metview family icon Vector.

list *colour_polar_field(...)*

Generates a vector field from a pair of intensity and direction fields, using arrows as the visual definition, but coloured according to the value of a third field (e.g. temperature). Corresponds to one of the options of the Metview family icon Vector.

list *curve(...)*

Generates an XY plot from two lists of numbers (X and Y values). Includes four types of plot - curve (curve or scatter plot), bar (bars between two values), area (filled area within two curves) and graph (charts with a number of the previous types together); these correspond to the options of the Metview family icon Curve.

definition *pm_grib_tephi(...)*

Returns a set input parameters (data, line colour, ground point coordinates) for a tephigram plot of GRIB (model output) data. Corresponds to one of the options of the Metview family icon Tephigram Input.

definition *pm_bufrtephi(...)*

Returns a set of input parameters (data, line colour, ground point coordinates) for a tephigram plot of upper air BUFR (observation) data. Corresponds to one of the options of the Metview family icon Tephigram Input.

definition *pm_pre1995gribtephi(...)*

Returns a set of input parameters (data, line colour, ground point coordinates) for a tephigram plot of pre-1995 GRIB (model output) data. Corresponds to one of the options of the Metview family icon Tephigram Input.

definition *budget(...)*

Computes a user defined meteorological budget. Four types available : Atmospheric energy, surface energy, hydrological, top of atmosphere radiation budget. You may specify an area and inclusion/exclusion of land/sea points. Corresponds to the Metview icon Budget.

definition *spec_graph(...)*

Generates a plot of spectra of a parameter as a function of Legendre polynomial order. Two types available - XY graph of amplitude against order or contour plot of amplitude against forecast length. Corresponds to the Metview icon Spectra.

list *scores(...)*

Computes scores - either rms error or correlation between forecast and analysis data. Corresponds to the Metview icon Scores.

error *datacoverage(...)*

Returns information on data coverage and quality of user specified observations. Corresponds to the Metview icon Data Coverage.

Visualisation icon-functions

list *plot_superpage(...)*

Plotting specification (view) for 2D geographical plots (maps). Corresponds to the Metview icon Map View.

definition *mapview(...)*

Plotting specification (view) for 2D geographical plots (maps). Returned variable should be provided as input to the view parameter of the `plot_page()` icon function. Corresponds to the Metview icon Map View.

definition *xsectview*(...)

Plotting specification (view) for 2D cross section plots. Returned variable should be provided as input to the `view` parameter of the `plot_page()` icon function. Corresponds to the Metview icon Cross-Section View.

definition *averageview*(...)

Plotting specification (view) for 2D average cross section plots. Returned variable should be provided as input to the `view` parameter of the `plot_page()` icon function. Corresponds to the Metview icon Average View.

definition *hovmoellerview*(...)

Plotting specification (view) for Hovmøller plots. Returned variable should be provided as input to the `view` parameter of the `plot_page()` icon function. Corresponds to the Metview icon Hovmøller View.

definition *tephigramview*(...)

Plotting specification (view) for tephigram plots. Returned variable should be provided as input to the `view` parameter of the `plot_page()` icon function. Corresponds to the Metview icon Tephigram View.

definition *vertprofview*(...)

Plotting specification (view) for 2D geographical plots (maps). Returned variable should be provided as input to the `view` parameter of the `plot_page()` icon function. Corresponds to the Metview icon Vertical Profile View.

definition *curveview*(...)

Plotting specification (view) for 2D geographical plots (maps). Returned variable should be provided as input to the `view` parameter of the `plot_page()` icon function. Corresponds to the Metview icon Curve View.

definition *satelliteview*(...)

Plotting specification (view) for satellite data. Returned variable should be provided as input to the `view` parameter of the `plot_page()` icon function. Corresponds to the Metview icon Curve View.

definition *emptyview*(...)

Plotting specification (view) for creating an empty area. Returned variable should be provided as input to the `view` parameter of the `plot_page()` icon function. Corresponds to the Metview icon Empty View.

Vis5D icon-functions

grib_to_vis5d(...)

Converts a GRIB to Vis5D format and launches the Vis5D GUI. Corresponds to the Metview icon Grib to Vis5D.

definition vis5d_object(...)

Defines a Vis5D plotting object (vertical slice, isosurface, etc.). Corresponds to the Metview icon Vis5D Object.

definition vis5d_trajectory(...)

Defines a Vis5D trajectory plotting object. Corresponds to the Metview icon Vis5D Trajectory.

definition vis5d_wind_object(...)

Defines a Vis5D wind plotting object (wind arrow, streamline, etc.). Corresponds to the Metview icon Vis5D Wind Object.

vis5d_window(...)

Defines and launches a Vis5D plotting window (area limits, grid, auxiliary elements). Corresponds to the Metview icon Vis5D Window.

Visual definition icon-functions

definition drawing_priority(...)

Metview specification of the order of visual definition plotting. The classic application is to plot a coastline element (land or sea shade) last, so as to mask out of a visualisation data over land or sea. Corresponds to the Metview icon Drawing Priority

definition legendentry(...)

MAGICS visual attributes for plot legend. Corresponds to the Metview icon Legend Entry

definition overlay_control(...)

Metview specification to control the overplotting of meteorological fields according to some of their attributes. Corresponds to the Metview icon Overlay Control

definition paxis(...)

MAGICS visual attributes for plot axis. Corresponds to the Metview icon Axis

definition pcoast(...)

MAGICS visual attributes for plotting coastlines. Corresponds to the Metview icon Coastlines

definition *pcont*(...)

MAGICS visual attributes for contouring and shading. Corresponds to the Metview icon Contour

definition *pgraph*(...)

MAGICS visual attributes for plotting graphs. Corresponds to the Metview icon Graph

definition *pimport*(...)

MAGICS visual attributes to control import of graphics files (jpeg, png) into a visualisation (by means of an import view). Corresponds to the Metview icon Import Plot

definition *pmimage*(...)

MAGICS visual attributes for manipulation of satellite images display. Corresponds to the Metview icon New Image Table

definition *pobs*(...)

MAGICS visual attributes for plotting observations. Corresponds to the Metview icon Observation Plot

definition *psymb*(...)

MAGICS visual attributes for plotting symbols. Corresponds to the Metview icon Symbol

definition *ptext*(...)

MAGICS visual attributes for text in plots. Corresponds to the Metview icon Text

definition *pwind*(...)

MAGICS visual attributes for plotting winds. Corresponds to the Metview icon Wind

UNIX Interfacing Functions

any *getenv*(string)

Returns the value of an environment variable given its name as the argument.

string *getenv*(string, number)

Returns the value of an environment variable given its name as the argument. If a second argument (number) is given and the number is zero, the function returns a string, even if the environment variable content looks like a date or a number.

none *nice*(number)

Lower the priority of the macro by calling the `nice()` system call.

string *putenv*(string, string)

Sets the value of an environment variable, given its name as the first argument and its value as the second argument.

number *shell*(...)

Returns the exit status of the command invoked.

none *sleep*(number)

Stops the macro for a given number of seconds

Macro System Functions

none *fail*(string)

Stops the execution of a macro, printing the input string to the main UI output area. Assigns error status to the macro icon (name turns red). Use to exit a macro on an error condition - input string should be a suitable error message.

any *fetch*(string)

Retrieves an item stored in a cache under the name specified as the argument.

```
s = fetch("wind speed")
```

The `fetch()` function returns `nil` if the specified data is not in the cache.

string *name*()

This function returns the name of the macro being executed. It can be used in conjunction with the `store()` and `fetch()` functions.

string *runmode*()

Returns the macro run mode - Execute, Visualise, Save, Examine, Edit, Batch, Prepare - as a string.

number *runmode*(string)

Returns 1 if the macro run mode is the same as the one specified in the input string and 0 if not

none *stop*(string)

Stops the execution of a macro, printing the input string to the main UI output area. Assigns OK status to the macro icon (name turns green). Use to exit a macro upon some non error condition - input string should be a suitable exit status message.

any *store*(string,any)

Saves the item given as the second argument in a cache under the name specified as the first argument

```
store("wind speed",s)
```


III- ICON REFERENCE

INTRODUCTION

This volume presents a description of all the Metview icons. Each icon is described in turn, with the background information presented first, followed by a description of the contents of the icon's editor window - for each input parameter the manual provides its significance, range of values/options and (as far as possible) details on the relationship with and dependencies on other parameters; finally a list of the actions that users can carry out on the icon and their outcomes is presented.

There are a variety of icons available in Metview and they can be classified in a variety of ways depending on the criteria users prefer. The Metview Desktop reflects this by storing the icon templates in customisable icon drawers - the default arrangement of icons grouped according to the type of work they are involved in can be changed by users to any other type of arrangement. You will note that a few icons are present in more than one drawer (the Observation Filter icon is present in the Filters drawers and in the Observations drawer), reflecting the multiple nature of some icons.

In this volume the Metview icons are presented in a sequence roughly similar to the arrangement of icon templates in the Metview Desktop drawers.

Work in Metview is carried out via "actions" on icons. Actions include *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate*, *Delete* and *Output*. The actions *Edit*, *Duplicate*, *Delete* and *Output* always lead to the same outcome, respectively :

- open the icon editor
- create a copy of the icon
- move the icon to the Wastebasket
- open the icon message window; all output (results and error messages) from the execution of the icon is piped into this window.

For more detail see "Operations on Icons" on page I- 24. The other actions *Execute*, *Visualise*, *Examine* and *Save* lead to different outcomes depending on which icon they are applied to. Hence, these are referred to in detail in each icon description. For details on handling and creating icons and working with icon editor window input tools see "Metview Icons" on page I- 17 and "Icon Editors" on page I- 34.

FOLDER



Folder is the Metview icon equivalent of a directory. Folder icons open as desktops which can contain any type of icon, including other folders.

Folder icons have no icon editor - they are created directly from the Basic icon drawer - see "Creating A New Desktop" on page I- 13.

You can also create symbolic links either through a Metview desktop menu option or externally through the `ln -s` command - see "Creating Links" on page I- 20. Symbolic links pointing to directories also appear as folder icons (with the names in italic). Depending on how permissions are set, a symbolic link folder may be read-only. Such folders appear with a little padlock on the icon picture.

There is no macro language equivalent.

Actions on the Folder icon

The actions available for the Folder icon are *Edit*, *Duplicate* and *Delete*. The first opens the folder into a desktop, the other two are self-explanatory.

NOTES



The Notes icon is a plain text editor. Use it (as the name implies) to write text notes about whatever you need. Any ASCII file brought into the `metview` directory (or one of its subdirectories) is assigned a Notes icon by Metview.

Notes icons are created from the Basic icon drawer - see "Creating Icons Explicitly" on page I- 18.

There is no macro language equivalent - use the `read()` function to the contents of a text file - see "General ASCII input" on page II- 34.

The Notes Editor

The Notes icon is a plain text editor. The section "Working with Text Icon Editors" on page I- 51 provides full details on how to work with the Metview simple text editor, including how to use another text editor (`vi`, `emacs`, ...) in its place.

SHELL SCRIPT



The Shell icon, like the Notes editor is also a plain text editor. It is used to write UNIX shell scripts which can then be executed from within Metview or Metview Macro. Any ASCII file beginning with `#!/bin/ksh` (or equivalent) brought into the `metview` directory (or one of its subdirectories) is automatically assigned a Shell icon.

Shell icons are created from the Basic icon drawer - see "Creating Icons Explicitly" on page I- 18.

The macro language equivalent is `shell()`

The Shell Editor

The Shell icon is a plain text editor. The section "Working with Text Icon Editors" on page I- 51 provides full details on how to work with the Metview simple text editor, including how to use another text editor in its place.

Actions on the Shell icon

The actions available for the Shell icon are *Execute*, *Edit*, *Duplicate*, *Delete* and *Output*. Output option becomes available if the shell script generates some text output or messages.

Execute - this action executes the UNIX shell script contained in the icon. Text output of the shell script and any error messages resulting from its execution are piped to both the icon's and the general Metview message area (File/Messages...)

Output - this action opens the icon message window; all output from the execution of the icon is piped into this window.

WEB ACCESS



This icon allows access to web pages from within the Metview environment.

The Web Access Editor

The editor contains a single input parameter, that of the URL you want to visit.

Url

Enter the URL you want to visit, e.g. `http://www.ecmwf.int`.

Actions on the Web Access Icon

The actions available for the Web Access icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate* and *Delete*. Those requiring some information are :

Execute, *Visualise*, *Examine*, *Save* - these actions all lead to the same result, the visualisation of the specified URL in Netscape.

DISPLAY WINDOW



This icon is particular among Metview icons in that it has a purely graphical user interface. It is used as a visualisation layout designer - a full description of its functionality is presented in "The Display Layout" on page I- 71.

MARS RETRIEVAL



The MARS Retrieval icon carries out the retrieval of data from the ECMWF archive. It is therefore the crucial, and necessarily the most used Metview icon. Data retrieved by this icon is used as input to most other Metview icons (though they may also use data from other icons). The Metview MARS Retrieval icon essentially represents a translation of the MARS (Meteorological Archival and Retrieval System) language into a Metview icon editor listing.

This section provides just the details of MARS that you require to use the Metview MARS Retrieval icon efficiently. It is not possible to cover in this User Guide all the possible combinations of parameters available in the MARS Retrieval editor, but the information provided should help you become more proficient in the preparation of data requests. Detailed information on MARS, and the availability and format of the archived data, can be found in the MARS User Guide, available at

<http://www.ecmwf.int/publications/manuals/mars/>

The macro language icon function is the `retrieve()` function.

Brief Overview of MARS

Data Formats

Archived data are stored in two formats, GRIB and BUFR. Detailed information on these two formats can be found in the following references :

WMO Manual on Codes (1995)

Volume I.1	A - Alphanumeric Codes
Volume I.2	B - Binary Code
	C - Common Features to Binary and Alphanumeric Codes
Volume II	Regional Codes and National Coding Practices

GRIB (*GRId in Binary*) - WMO defined format for meteorological field data, or (more generally) any regularly spaced gridded data. All ECMWF model output is in GRIB format.

BUFR (*Binary Universal Form Representation*) - WMO defined format for point data (irregularly spaced). Archived observations are in BUFR format.

Contents of MARS

Here we provide an overview of the meteorological data contained in the archive of the ECMWF. The data can be grouped in the following categories :

Operational Data

Atmospheric model

Analysis and Forecast

- Analysis (synoptic hours 00, 06, 12, 18 UTC) :
 - Surface fields
 - Model Level Fields (1 to 60, LNSP is level 1)
 - Pressure Level fields (15 levels)
- Forecast (10 day forecast based on 12 UTC analysis)
 - Surface, Model level and Pressure level
 - 3-hourly interval from 00 to 72 hours
 - 6-hourly interval from 72 to 240 hours

This is the situation for data produced from January 1999 - older data may not exist at such fine temporal resolutions, e.g. back to November 1990 data exists as 3 hour data until 12 hours, 6 hour data until 120 hours and 12 hour data until 240 hours.

Wave Model

Analysis and Forecast

- Analysis (synoptic hours 00, 06, 12, 18 UTC) :
- Forecast (10 day forecast based on 12 UTC analysis)
 - 3-hourly interval from 00 to 72 hours
 - 6-hourly interval from 72 to 240 hours

This is the situation for data produced from October 2000 - older data may not exist at such fine temporal resolutions, e.g. 3 hourly data did not exist before and 6 hourly data extended to 120 and then 240 hours.

EPS, Ensemble Prediction System

Contains the same elements as the operational archive but for different forecasts from perturbed initial conditions :

- Control forecast (operational forecast at T159/T255)
- Perturbed forecast (1 to 50, 31 prior to December 1996)
- Forecast probabilities
- Forecast mean probabilities
- Ensemble mean forecast
- Cluster products (6 clusters)
- Tubes (another type of cluster products)

Observations

- Surface data
- Vertical soundings
- Upper-air data

- Satellite images

Boundary Conditions

Also known as short cut-off . Four additional Analysis are run for 00, 06, 12 and 18 UTC with a cut-off time of 4 hours, followed by global 4-day forecast to provide some Member States with boundary conditions for their limited area models. All Analysis data but only the forecast from 00 UTC are archived.

Multi-Analysis Ensemble

Every day ECMWF receives Analyses from NCEP, UK Met Office, MeteoFrance and DWD and runs 5 forecasts, 1 based on each different analysis plus one compound of all the analyses (consensus) including ECMWF's Analysis.

Seasonal Forecast

ECMWF started an experimental programme in seasonal prediction in 1995, which attempts to predict seasonal changes by coupling three models: atmospheric , wave and ocean models.

Monthly Forecast

This project has the same setup as the seasonal forecast and runs every week or two producing forecasts of one month in length, coupling atmospheric, wave and ocean models.

Special Projects

ERA-15 / ERA-40

The ECMWF Re-Analysis (ERA) project has produced a new, validated 15 year data set of assimilated data for the period 1979 to 1994. This has been named ERA-15, and available datasets include: analysis, forecast and forecast accumulations as output from atmospheric model, as well as analysis and forecast from a wave model re-analysis. There is also a Monthly Means data sets contain data at the resolution of the data assimilation and forecast system used by ERA-15. At the time of writing (2002), another Re-Analysis is being carried out, covering the period mid-1957 to 2001. This has been named ERA-40.

DEMETER

Acronym of the EU-funded project entitled **D**evelopment of a **E**uropean **M**ulti model **E**nsemble system for seasonal to **i**n**T**ERannual prediction. The objective of the project is to develop a well-validated European coupled multi-model ensemble forecast system for reliable seasonal to inter annual prediction. Seven comprehensive European global coupled atmosphere-ocean models are being installed at ECMWF : ECMWF, MeteoFrance, LODYC, Met Office, MPI, INGV and INM-HIR-LAM.

PROVOST

Acronym of **P**Rediction **O**f climate **V**ariations **O**n **S**easonal to inter annual **T**ime scales. They are a set of experiments from four centres - ECMWF, MétéoFrance, EDF and UK Met Office. The experiments are 120 day runs from 9 consecutive starting days, with write-ups every 24 hours of Pressure level and Surface data.

ECSN-HIRETYCS

ECSN is the European Climate Support Network. **HIRETYCS** is the **HIgh REsolution Ten Year Climate Simulation**. This data set consist of 10-year climate simulation produced at three centres - Centre National de Recherches Météorologiques (CNRM), Max Planck Institute (MPI) and UK Met Office.

Research and Member State's Experiments

A vast amount of data is archived daily containing experiments produced by ECMWF's Research Department or experiments run by Member States at ECMWF. Basically, an experiment can address any of the areas of Meteorology and it is archived accordingly.

Monthly and Climatology Data

ECMWF maintains an archive of monthly means data from the atmospheric and wave model archive.

- Analysis and Forecasts, Atmospheric and Wave models

Data from Other Centres

Subsets of products (analysis and forecast) from other centre's model output (missing data is replaced by dummy fields) :

- Bracknell
- Washington (plus ensemble forecast)
- Offenbach
- Paris
- Tokyo
- Montreal

There are other fields which are used as input for the ECMWF forecasting system (e.g. Sea Surface Temperatures from Washington which are used by ECMWF's Analysis).

Spatial Coordinate Systems

GRIB data is archived in one of the following spatial coordinate systems :

- Spherical Harmonics (SH) - Upper air fields.
- Gaussian Grid (GG) - Surface fields.
- Latitude/Longitude (LL) - other centres, surface fields, wave data.

The correspondence between the three types of grid resolutions is as follows :

Spectral	Gaussian	Lat/Lon
T63	N48	1.875
T95	N48	1.875
T106	N80	1.125
T159	N80	1.125
T213	N160	0.5625
T255	N128	0.7

T319	N160	0.5625
T399	N200	0.450
T511	N256	0.351
T639	N320	0.28125
T799	N400	0.225
T1023	N512	0.176
T1279	N640	0.141
T2047	N1024	0.088

See http://www.ecmwf.int/products/data/technical/gaussian/spatial_representations.html for a complete up-to-date version of this table.

You can reduce the resolution of the retrieved data and/or convert from one representation to another. Note that the choice of resolution of the retrieved data may have important implications for the efficiency of the task you need to carry out - see "Postprocessing" below.

Postprocessing

MARS can carry out some postprocessing of the retrieved data for you. Available postprocessing includes :

- Coordinate Conversions
 - SH to : SH (different resolution through triangular truncation), GG, LL
 - GG to : GG (different resolution), LL
 - LL to : LL (different resolution)
- Extractions
 - Sub-areas, either in GG or LL representation
- Derived Fields
 - Wind components (U/V) from VO (Vorticity) and D (Divergence)

Depending on what you want to do you may be able to avoid unnecessary increases in the processing time and memory requirements (see "Retrieval Efficiency" below).

Retrieval Efficiency

If your requests are correctly specified you do not need to know or be aware of the internal workings of the data retrieval. However, the details below may help you improve the efficiency of your requests and understand why some requests may sometimes go wrong or take longer than you would expect/like.

MARS data can be retrieved by a Metview user either from the automated tape archive or from one of the data caches. The data caches hold the most recent analysis and forecasts plus any data which other users may have recently retrieved. You will find that if your data is already in a cache the retrieval time will be significantly shorter than if it has to be retrieved from the archive. Any data you retrieve is also kept in your own private cache (on your local disk). This cache is deleted when you quit Metview.

When carrying out retrievals of data, try to estimate how many fields you are actually retrieving in a single request. Retrievals of a few thousand fields (which can arise from a deceptively simple request) are best carried out with the MARS interactive language, outside Metview.

If you carry out some post-processing on retrieved data (e.g. SH to LL coordinate conversion, extraction of sub-area) the post-processing is carried out on your local disk, using your workstation's CPU.

The most frequent post-processing work is the conversion of data from spectral representation to a lat/long grid. When doing so, it is recommended that you specify a lower resolution in the **Resol** parameter and the required lat/long grid dimensions (in long/lat order) in the **Grid** parameter. In this way the data is *first truncated* to the lower resolution, *and then converted* to the required lat/long grid, and this may lead to important savings in visualisation time and used resources, particularly if you want to visualise data originally in high resolution (T213 and above) in a coarse lat/long grid.

The recommended course of action is to identify the coarsest lat/long grid mesh size that is compatible with your requirements and then select the corresponding truncation. The advantages of this procedure are such that as of May 1998 it has been built in the MARS Retrievals. So when you request data on a lat-long grid, by default the data is first truncated to a lower resolution according to the table below (see parameters "Resol" on page III- 25 and "Grid" on page III- 26)

Grid Increment	Truncation(SH/GG)
2.5 <= i	T63 / N48
1.5 <= i < 2.5	T106 / N80
0.6 <= i < 1.5	T213 / N160
0.4 <= i < 0.6	T319 / N160
0.3 <= i < 0.4	T511 / N256
0.15 <= i < 0.3	T799 / N400
0.09 <= i < 0.15	T1279 / N640
0.0 <= i < 0.09	T2047 / N1024

See http://www.ecmwf.int/publications/manuals/mars/guide/Post_processing.html for a complete up-to-date version of this table.

If visualising fields in spectral representation (without converting to a lat/long grid), the automatic truncation is not in place, but you should use the truncation (**Resol**) with the lowest T number that still gives you a suitable visualisation; quite frequently high resolution fields can be truncated prior to visualisation without appreciable loss of information.

The MARS Retrieval Editor

The parameters present in the MARS Retrieval editor window are identical to those of the MARS language and the syntax for their specification is the same. If you are conversant with the MARS language you should have little problem dealing with the Metview MARS Retrieval editor. Otherwise, a description of the parameters and their syntax is provided below. The parameters present in the MARS Retrieval editor are only a subset of the full complement of parameters of the MARS language.

In common with other Metview icon editors, the MARS Retrieval editor has some automation built in. This means that specifying a value for a given parameter may force the adoption of particular settings for another parameter : for example, selecting **Surface** for **Levtype** disables **Levelist**. Note that reversing the setting for the "controlling" parameter may not reverse the settings of the dependent parameters.

The full list of parameters in the editor follows :

Class

Archived data may arise from routine **Operations** (od), from **Research** (rd) experiments, **Reanalysis** data sets (er, e4), **Ecsn** (European Climate Network Support) (cs) and Member States (2 letter country ISO code, e.g. uk, de, es, pt, fi, ...). **Operations** (default) and **Research** are the most frequently used data classes

Type

Identifies the type of data required. There are 45 types available, some of which may no longer be produced (e.g. **Initialised Analysis**). Available types include **Analysis** (default), **First Guess**, **Forecast**, **Observations**, satellite **Images** and **Ensemble** products, amongst others. Some types (e.g. **Observations**) when selected, will disable some parameters (**Levelist**), enable others (**Obstype**) or force others to some value (**Repres** to **Bufr**)

Also note that what you choose for **Type** may impose restrictions on the valid choices for other parameters (particularly for **Param** and **Level**) : e.g. if **Type** is set to **Sensitivity Gradient**, **Param** is restricted to T, VO, D or Lnsp. If **Type** is set to one of **Ensemble Mean**, **Ensemble Standard Deviation**, **Tube**, **Cluster Mean** and **Cluster Standard Deviation**, **Param** is restricted to z (at **Level** 1000mb and 500mb) and T (at **Level** 850mb and 500mb). Note that these restrictions are not enforced within the MARS Retrieval editor.

Stream

Identifies the origin of the data or the Project that generated the data. This parameter selects from a large variety of streams, such as the **Daily Archive** (default), data from other centres, **Ensemble Forecasts**, **Monthly** means, etc

Expver

Specifies a code for data files arising from a given experiment. Each experiment is awarded a unique code. Production data is assigned a 1 or 2, Operations experiments 11, 12, ..., Research (or MS) experiments four letter codes. Users wanting to retrieve Research experiments need to know in advance the code for the specific experiment and its nature. For this information contact User Support.

Repres

Specifies the representation, i.e. **Spherical Harmonics**, **Gaussian Grid**, **LatLong Grid**, **Bufr**, **Space View** (satellite images) or **Ocean Grid**. If **Type** set to **Observations**, choose **Bufr** or **Space View**. See also parameters **Resol** and **Grid**.

*Note : **Resol** is purely a required identifying parameter and does not act on the retrieved data - it is not the parameter to use to reproject data archived in SH (say) as an LL grid (see "Grid" below)*

Obstype

Specifies the observation type and subtypes. Use the editor help drop down list to see all available options. Only available if **Type** set to **Observations**

Levtype

Specifies the type of levels of the retrieved data. Available options include **Pressure Levels** (default), **Surface**, **Model Levels**, **Depth**, **Potential Vorticity** and **Potential Temperature**. Choosing **Surface** disables **Levelist**.

If **Type** is set to **Sensitivity Gradient**, **Levtype** must be set to **Model Levels**

Levelist

Lists the required pressure or model levels, specified as text. Separate each value by a forward slash - e.g. 1000/850/700/500/400/300. Unavailable if **Levtype** is set to **Surface**. Specify pressure levels in descending order of magnitude, ascending for model levels

Param

Specifies the required meteorological parameter(s) to retrieve. You may specify (part of) its name(s) or an abbreviation or code number. Use the editor help drop down list to see all available options. If specifying part of the name, you need to make it unique; ambiguity will give you an error message

Date

Specifies the analysis date or forecast basedate. Allowed formats are :

- Absolute as YYYY-MM-DD, YYYYMMDD (set DD to 00 for monthly means data). You can also use Julian days : YYYY-DDD.
- Relative as -n; n is the number of days before today (-1 = yesterday)
- Name of month (e.g. Jan) for **Type** set to **Climatology**
- 00 (zero) to retrieve Operational monthly means

Dates should be specified with four digit years - a warning message appears otherwise. To retrieve observations, **Date** must be set to -2 or older, as there is a lag of a couple of days before observations are archived. For multiple date specification :

- To specify a number of different dates use : YYYYMMDD1/YYYYMMDD2/...
- To specify a range of dates use : YYYYMMDD1/t0/YYYYMMDD2/by/INCR; the default increment is 1

Fcmonth

Labels the complete calendar months which follow the forecast start date. For example, **Fcmonth**=2 is the second *whole* calendar month *after* the forecast start. For a forecast starting on 1 Jan 2000 this would be Feb 2000. For a forecast starting on 2 Jan 2000, this would be Mar 2000.

Fcperiod

For a stream containing variable period forecast means such as **Stream**=mofm, **Fcperiod** defines the period over which the field has been averaged, and it is given instead of **Step**. **Fcperiod** is given in units of days, in the format nnn-mmm, where nnn is the first day of the average and mmm is the last day of the average. For example, **Fcperiod**=04-10 represents the average of days 4 to 10 of the forecast.

Time

Specifies the time for observations, analysis, forecast base time or first guess verification as HHMM. Default is 1200. If MM is omitted it defaults to 00. For multiple time specification :

- To specify a number of different times specify HHMM1/HHMM2/...
- To specify a range of times use : HHMM1/t0/HHMM2/by/INCR; the default increment is 6 hrs. **NOTE** - this construct *will not work* for observations. Observations require you to specify all the time steps you need, explicitly

Also, please note that observations are organised in daily files containing data from 00:01:00 to 24:00:00. This means that if trying to retrieve time steps 00, 06, 12, 18 for a given date, you have to specify two requests, one for (date-1), **Time**=24, and another for (date), **Time**=06/12/18. If you are dealing with fields, you only need a single request with **Time**=00/t0/18/by/06.

Range

Specifies a time range in minutes for observations counting from the time specified in **Time**. This enables you to retrieve observations data within a tolerance around a specified time - e.g. `time=2100 range=360`, denotes observations from 21:00 to 03:00 next day.

For fields it is mostly used for ocean data. For a timeseries product, **Range** defines the values in time over which the time series extends. For a time averaged product (**Product**=`tavg`), **Range** is the length of the period over which the averaging has been made. **Range** is generally used with **Step**, which defines the end point of the relevant time interval. For example, for type forecast (**Type**=`fc`), **Step** gives the number of hours at the end of the timeseries, and range gives the number of hours between the beginning and the end of the time-series.

Step

Specifies the forecast time step in hours (HH) from forecast base time, or length of forecast which verifies at first guess time. Default is 00, which must be used if the data retrieved is not a forecast or first guess or derived from forecasts or first guesses

- To specify a number of different steps specify `STEP1/STEP2/...`
- To specify a range of steps use : `STEP1/TO/STEP2/BY/INCR`

Some choices for other parameters impose restrictions on values that **Step** may take :

- If **Type** is set to **Sensitivity Forecast**, **Step** is restricted to values of 0, 12, 36, 48
- If **Type** is set to **Sensitivity Gradient**, **Step** is restricted to 0, 24, 48 if **Iteration** is 0 and restricted to 0 if **Iteration** is one of 1, 2 or 3

Reference

Specifies the reference forecast timestep (in hours) at which the assignment of ensemble forecast members to tubes is carried out. This parameter requires **Type** set to **Tube** and its allowed values are 96, 144, 168, 192 and 240, the reference forecast timesteps for tubing currently in use.

The tubing categorisation made at the reference forecast timesteps is applied to a number of preceding forecast timesteps. These differ according to the reference forecast time step. The table below shows the values you can specify in **Step** for each of the allowed values of **Reference**, or conversely, the values you can specify in **Reference** for a given value of **Step** :

Reference	Step
96	48, 72, 96
144	96, 120, 144
168	72, 96, 120, 144, 168
192	144, 168, 192
240	192, 216, 240

Number

Specifies the number to be used in EPS products. Which number you specify here depends on which ensemble parameter you want to retrieve :

If retrieving Ensemble Perturbed Forecasts you must set :

- **Type** to **Perturbed Forecast**

- **Stream to Ensemble Forecasts**

In this case, **Number** specifies the EPS (Ensemble Prediction System) number and can take values from 1 to 50 (to 32 prior to 10/12/1996). Either specify :

- n1/TO/n2 to retrieve all ensemble forecasts from n1 to n2
- n1/n2/ . . . (ascending order), to retrieve individual ensemble members

If retrieving Cluster Products you must set :

- **Type to Cluster Mean or Cluster Standard Deviation**
- **Stream to Ensemble Forecasts**

In this case, **Number** specifies the number of the cluster(s) of ensemble forecasts for which a mean or a standard deviation can be obtained. It can take values from 1 to 6 : Either specify :

- n1/TO/n2 to retrieve all clusters from n1 to n2
- n1/n2/ . . . (ascending order), to retrieve individual clusters

If retrieving Ensemble Forecast Probabilities you must set :

- **Type to Forecast Probability**
- **Stream to Ensemble Forecasts**

In this case, **Number** specifies the number of the ensemble forecast probability. It can take values from 1 to 4 for T850 and Precipitation and 1 or 2 for 10m wind speed (for details on the significance of these numbers see MARS User Guide (For Data Retrieval) - ECMWF Computer Bulletin B6.7/2, pages 133-135). To specify a range use n1/n2 to retrieve probabilities n1 to n2

If retrieving Tube Products you must set :

- **Type to Tube**
- **Stream to Ensemble Forecasts**

In this case, **Number** specifies the number of the tube. It can take values from 0 to 9. Tube 0 is the central cluster which is the mean of a number of ensemble members which are the nearest to the mean of the whole ensemble. The remaining members are grouped into tubes numbered 1 to 9, each tube actually being the ensemble member representing an extreme case. Either specify :

- n1/TO/n2 to retrieve all tubes from n1 to n2
- n1/n2/ . . . (ascending order), to retrieve individual tubes

Domain

This parameter is used in the specification of Cluster, Tube and Wave data. Only a subset of the available options should be considered for each type of retrieved data :

- for Cluster and Tube data, the working options for this parameter are **Globe, North West Europe, North East Europe, South West Europe** and **South East Europe**. Note that option **Globe** actually gives you Europe as it stands for *Global European Area*. For this data, **Domain** specifies the area over which the cluster computation is carried out, *it does not* specify a sub-area (see **Area** on page III-25 if you want to specify a sub-area) - the retrieved fields are still global in geographical scope.
- for Wave data the working options for this parameter are **Globe, Mediterranean, Northern Hemisphere** and **Southern Hemisphere**. For this data, **Domain** specifies the geographical

domain for which the data were produced - internally, wave data is archived in different geographical chunks, due to model output format and data management restrictions.

- **Fastex** - domain of the FASTEX experiment.

Frequency

Specifies the required frequency components of wave model spectral fields. These wave model fields are discretised in a number **NFRE** of frequencies and a number **NANG** of directions at each grid point. **NFRE** is currently 25. Specify numbers from 1 to **NFRE** :

- To specify a range use $n1/TO/n2$ to retrieve all directions numbered from $n1$ to $n2$ (maximum $n2$ is **NFRE**)
- To specify a list of directions provide a list of numbers separated by a slash : $n1/n2/n3/...$

This parameter requires :

- **Stream** set to **Wave**
- **Levtype** set to **Surface**
- **Repres** set to **LatLong Grid**
- **Parameter** set to 251

Direction

Specifies the required direction components of wave model spectral fields. These wave model fields are discretised in a number **NFRE** of frequencies and a number **NANG** of directions at each grid point. **NANG** is 12 for the global wave model and 24 for the mediterranean wave model. Specify numbers from 1 to **NANG** :

- To specify a range use $n1/TO/n2$ to retrieve all directions numbered from $n1$ to $n2$ (maximum $n2$ is **NANG**)
- To specify a list of directions provide a list of numbers separated by a slash : $n1/n2/n3/...$

This parameter requires :

- **Stream** set to **Wave**
- **Levtype** set to **Surface**
- **Repres** set to **LatLong Grid**
- **Parameter** set to 251

Diagnostic

Specifies the diagnostic function number code. This parameter requires :

- **Stream** set to **Sensitivity Forecasts**
- **Type** set to **Sensitivity Gradient** or **Sensitivity Forecasts**
- **Iteration** set to a suitable value (see **Iteration** below)

Diagnostic may take values of 0, 1, 3 or 5 :

0 - diagnostic function is the initial analysis or the control forecast. Not allowed if **Type** is set to **Sensitivity Gradient**.

1 - diagnostic function is the 2 day forecast error (2 day fc minus verifying analysis)

3 - diagnostic function is the difference between the 2 day forecast from day d , and the 1 day forecast from $d+1$. Not allowed if **Type** is set to **Sensitivity Forecast**

5 - diagnostic function is the difference between the 2 day forecast from day d at 12z, and the 36h forecast from d+1 at 00z

Iteration

Specifies the step number in the algorithm for the minimization of the diagnostic function. It may take values from 0 to 3. You must specify **Diagnostic** (see above) and set **Type** to one of **Sensitivity Gradient** or **Sensitivity Forecast**. What you specify for these parameters determines the values that **Iteration** may take as shown in the table below :

		Sensitivity Gradient	Sensitivity Forecast
Diagnostic	0	not allowed	0
	1	0 to 3	3
	3	0	not allowed
	5	0 to 3	3

Channel

Specifies the TOVS/ATOVS channel number of model fields transformed to brightness temperature : HIRS 1-20, MSU 21-24, SSU 25-27, AMSUA 28-42, AMSUB 43-47.

Requires **Parameter** set to **Brightness Temperature** (194), **Levtype** set to **Surface**, **Repres** set to **Gaussian Grid**. It is only available for **Step** set to 6 (for 6-hourly 4D-Var, i.e. before 20000912) and set to 12 (for 12-hourly 4D-Var), **Data Type** set to **Errors in First Guess**. This may change in future.

Ident

Specifies the satellite identification number (e.g. 52 is METEOSAT 5, 53 is METEOSAT 6). Use the assist button to obtain a window with information on the satellite images available in the archive for the current date. This information allows you to set values for **Ident** (satellite id number), **Time** (time of the image) and **Obstype** (number for the channel). The possible values for **Ident** and **Obstype** are also listed.

Origin

This specifies the point of origin of data from other institutions (e.g. NCEP's SST fields). Options are **Off**, **Washington**, **Bracknell**, **Fleet Numerical**, **Offenbach**, **ECMWF**, **Toulouse**, **Paris** or **Consensus**. **Stream** must be set to **Supplementary Data**.

System

Labels the version of the operational forecasting system. For example, the ECWMF seasonal forecasting system uses **System**=1 for the Cycle 15r8 based system used between 1997 and 2002, and **System**=2 for the Cycle 23r4 based system introduced in 2002. For research experiments, **System**=0.

Method

For a given system or experiment, labels how the analyses and forecasts were produced. **Method**=0 denotes the case when no ocean data assimilation was used to prepare ocean initial conditions. **Method**=1 is the standard case of data assimilation. If a given system or experiment uses several different data assimilation systems, then there may also be methods 2,3 etc.

Product

For ocean fields, the type of product which is archived; in particular how the archived field is defined in time. The product may be an instantaneous field (**Product**=inst), a field accumulated in time from the start of the integration (**Product**=tacc), a field averaged in time over a given range (**Product**=tavg), or a timeseries (**Product**=tims).

Section

For ocean fields, the spatial orientation of the archived field. The field might be horizontal (**Section=h**), or, if vertically oriented, it might be a meridional (**Section=m**) or zonal (**Section=z**) section or timeseries, or it might be a vertically oriented timeseries (**Section=v**).

Latitude

For a zonally oriented field or time-series, specifies the latitude location in degrees, positive values for north.

Longitude

For a meridionally oriented field or time-series, specifies the longitude location in degrees east, in the range 0-360.

Resol

Specifies the triangular truncation of retrieved data archived in SH. If the data to be retrieved is not archived as SH, you needn't change this parameter. The choice of truncation has important implications for the efficiency of the task to be carried out - see "Retrieval Efficiency" on page III- 17.

The default is **AUTO** which stands for automatic truncation. With automatic truncation, the lowest resolution compatible with the value specified in **Grid** is automatically selected for the retrieval :

Grid Increment	Truncation(SH/GG)
2.8125000 and greater	T63 / N32
1.8750000 to 2.8124999	T106 / N48
1.4062500 to 1.8749999	T127 / N64
1.1250000 to 1.4062499	T213 / N80
0.7031250 to 1.1249999	T255 / N128
0.5625000 to 0.7031249	T319 / N160
0.3515625 to 0.5624999	T511 / N256
less than 0.3515625	T639 / N320

If you need to reduce the resolution of the data (without converting to LatLong grid) set **Grid** to **OFF** and **Resol** to a T number like the ones above - if you leave **Resol** set to **AUTO**, the data is retrieved in the resolution in which it was archived. With **Resol** set to **AV** the data is retrieved in the resolution in which it was archived irrespective of what you set **Grid** to.

You can specify your choice via the assist button which has a check-list of the most common T numbers and the **AUTO** and **AV** options.

Accuracy

Specifies the number of bits per value to be used in the generated GRIB coded fields. A positive integer may be given to specify the preferred number of bits per packed value. This must not be greater than the number of bits normally used for a Fortran integer on the processor handling the request (typically 32 or 64). As an alternative to explicitly specifying a number, you may also specify one of three predefined character codes:

- N - Normal accuracy (default - retrieves data at stored accuracy)
- R - Reduced accuracy
- L - Low accuracy

Area

Specifies the subarea of data to be extracted. If set to **GLOBAL** (default) the full globe is retrieved. The editor offers some predefined areas (Europe, and four quadrants for each hemisphere).

Alternatively, you may specify an area of your own, by inputting the area limits as North/West/South/East; if South > North, the values are swapped and a warning issued; southern latitudes and western longitudes must be given as negative numbers. Example : Europe may be defined by 75/-20/10/60 = 75N to 10N, 20W to 60E. Areas can be defined to wrap around the globe, e.g., in order to retrieve all longitudes but Europe one would specify 75/60/10/-20.

If you want to specify a subarea, you *must* retrieve your data as a Lat/Long grid or as regular Gaussian grid (using **Grid/Gaussian**), otherwise it defaults back to GLOBAL. If the area requested is inconsistent with the specified grid mesh, the area is adjusted as necessary to ensure consistency and you are warned by a message in the Metview message area. The required interpolation is carried out on your workstation.

Block

Specifies WMO Block number(s) for observations, e.g. 02

Rotation

Specifies a new position for the South Pole - MARS will rotate the grid positioning the South Pole to the coordinates given. Enter the coordinates of the new position of the South Pole as two numbers separated by slash, e.g. -30.0/10.0.

Frame

Specifies the number of points to be selected from a sub-area inwards. It works together with the parameter **Area**. For example, **Area**=europe, **Grid**=2/2, **Frame**=2 will select a frame with outer limits 74/60/10/-20 and inner limits 72/58/8/-18.

Bitmap

Specifies a UNIX filename containing a bitmap definition, in the format defined in *Appendix 3 of ECMWF Meteorological Bulletin M3.1: The dissemination of ECMWF products to Member States*, D.Jokic. revision 3 9/97

Grid

Specifies the output grid mesh. The default is OFF, to retrieve data without changing to another representation. When **Grid** is OFF, **Resol** is set to AV, though you can specify any T number you want. To retrieve data archived in Lat/Long without changing its resolution, set **Grid** to AV.

SH to GG - To change the representation from SH to GG, specify a single positive integer (one of the N numbers shown in "Data Formats" on page III- 13) - this is the number of latitude lines between the Pole and the Equator. The type of gaussian grid you require (reduced or regular) is defined in parameter **Gaussian** below.

SH/GG to LL - To change the representation from SH or GG to LL, specify two numbers (up to three decimal places) separated by a forward slash, e.g. 1.5/1.5, which specify the longitude and latitude grid mesh size; note the order: longitude then latitude. In this case, **Resol** is set to AUTO (automatic truncation) - see "Retrieval Efficiency" on page III- 17.

GG and LL resolution change - to simply change the resolution of data originally in GG or LL, while keeping it in the same representation, specify the new resolution as a single positive integer for Gaussian grid data (one of the N numbers shown in "Data Formats" on page III- 13) or a set of two numbers (longitude then latitude resolution) separated by a forward slash for LL data. For data originally in regular Gaussian grid, **Gaussian** must be set to **Off** or **Regular** (the result is the same), as **Reduced** is not allowed. For data originally in reduced Gaussian grid, set **Gaussian** to what you require.

Note : You cannot use just any number for the LL grid. In the first place, the minimum value you can use is 0.28125; anything smaller is ignored and reset to 0.28125. Secondly, you can only specify a rational number which is an integer fraction of 90 degrees (latitude range from the Equator to the Pole). E.g. 2.00, 2.25, 2.50, 2.8125, 3.00, 3.50, 3.75, 4.50, 5.00, etc.,

Gaussian

Set to one of **Reduced**, **Regular**, **Off**. This specifies the type of gaussian grid you get from the change in representation specified in **Grid** above.

Specification

Specification of reduced gaussian grid to use. Can only be used with N80 or N160 reduced gaussian grids to force an old grid definition to be used - e.g. for Research Department designated _12, which has 12 points in the northernmost and southernmost latitudes, **Specification** = 12.

Padding

For historical reasons, each GRIB message retrieved is padded to a multiple of 120 bytes, and each BUFR message to a multiple of 8 bytes. This parameter specifies a number of bytes to use for padding, e.g., **Padding**=0 for no padding.

Duplicates

Set to one of **Keep** or **Remove**. Only active if **Type** is set to **Observations**, or **Type** set to **Images** and **Stream** set to **Ssmi** (MW satellite soundings). The archive contains duplicated observations, which are removed or kept in the target file according to how you set this parameter.

Database

Specifies in which of the available databases the search for the required data should be carried out. This feature is meant to be used by MARS developers and its use is best avoided by the general user. There is no significant advantage in terms of speed of data retrieval to be gained by use of this parameter.

Expect

Specifies how many fields you are expecting to retrieve. Some datasets do not have all parameters at all levels. Use this parameter in case your request includes data gaps, e.g if a data set only had parameter 129 at 1000/500 and parameter 130 at 850, you would specify:

```
param = 129/130
levelist = 1000/850/500
expect = 3
```

to correctly retrieve the three existing fields.

If **expect = any** is provided, MARS will retrieve any fields or observations for the given request, but it will not fail if it does not find any data - the returned variable in this case will have type 'nil'. This can be useful when checking for unavailable data within a macro as the following example shows:

```
mars = retrieve
(
  expect   : 'any',
  levtype  : 'ml',
  levelist : 91,
  param    : 'lnsp'
)

print ( 'waiting for the retrieval' )
```

```
if type (mars) = 'nil' then print ('mars: no fields available')
else print ('mars: number of fields retrieved:', count(mars))
end if
```

Help in Specifying Retrievals

The large number and variety of data archived at ECMWF prevents an exhaustive listing of all the valid combinations of parameters (valid in the sense that they successfully retrieve data). The previous section offers some guidance on how to set each individual parameter and some of the links between parameters (in the sense that the setting of one parameter implies a particular setting for another parameter).

However, there is still plenty of scope for mis-specification of requests. For example from the previous section you know what to set for Type, Stream and Ensemble to retrieve an EPS Perturbed Forecast. But you do not know how far back in time these products go, which meteorological parameters they were produced for, their representation, etc.

MARS User Guide

The best source of help is the MARS User Guide in the ECMWF's web pages :

<http://www.ecmwf.int/publications/manuals/mars/>

This covers the details of the MARS language and it is therefore more extensive than the information in this manual.

MARS Retrieval Icon Templates

More specific help can be provided by the MARS Retrieval icon templates. These icons are associated with the icon editor and reside within the icon editor template drawer. By default you are provided with a set of MARS Retrieval templates and you can always add your own icons as time progresses. Templates are good at showing which sets of parameters go together for a particular retrieval - e.g. choosing the template **Satellite IR Images** immediately gives you the right specifications to retrieve satellite IR imagery.

Of course you need to find a template close to your requirements and templates can't tell you about changes in the availability of a given data or about changes in its specification.

For details on using icon templates, please see "Template Icons" on page I- 47.

WebMARS

ECMWF has a web based MARS interface which allows you to retrieve and query the MARS archive via a web browser. This has been configured in such a way that it allows you to build your request in a piecemeal fashion.

This means you can travel down the request tree branching out at each choice you make. This is in contrast with the MARS Retrieval editor where you need to specify all the elements and then sub-

mit them. For example to retrieve EPS forecasts of upper air fields, in WebMARS you could choose successively :

- operational system
- ensemble
- operational runs
- perturbed forecast
- year
- month
- levtype (press or sfc)
- day
- fc step, ensemble member number, level, parameter

At each branching you can check whether you're on the right track. When you get to the end and make your final choice of data, you have the following options :

- check for availability - check which fields of your request actually exist (e.g. some fields may not exist at all levels or forecast steps)
- view request in MARS language - you can check what your request looks like in MARS language (and get some idea of what to specify in the MARS Retrieval editor)
- evaluate cost of request - you can know the request in terms of N fields, X Kbytes on T tapes
- retrieve selection - actually retrieve the requested data
- plot selection - prepare a plot of your request

You can use WebMARS in the usual way. Metview provides a direct link from the MARS Retrieval editor to the WebMARS service - the MARS Retrieval Editor has a drawer with buttons which allow you to interface with the WebMARS service in a variety of ways :

- Launch WebMARS
- Check the existence of the MARS request
- Retrieve the MARS request

Actions on the MARS Retrieval Icon

The actions available for the MARS Retrieval icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate*, *Delete* and *Output*. Those requiring some information are :

Execute - this action carries out the retrieval of the data from the archives. Note that the retrieved data is stored in a cache on your workstation, but it is not automatically visualised. Apart from the changes of colour to the icon name, nothing else visible happens.

Visualise - this action carries out the visualisation of the data, using the default plot window. If the data has not been retrieved before, the visualise action also carries out the retrieval of the data into the cache. If the data is already in the cache, the visualisation happens much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62.

Examine - this action allows you to examine the contents of the data (to be) retrieved (see "Examining GRIB Data" below).

Save - this action allows you to save the retrieved data as a GRIB file (see "Saving MARS Retrieval Output (as Files)" below).

Saving MARS Retrieval Output (as Files)

To save the retrieved data to your local disk, use the **Save** option from the icon menu. If the data has not yet been retrieved (i.e. it is not in the cache) the *Save* action also carries out the retrieval, places the data in the cache and saves to a file. An output window (see Figure III-1) is launched which by default will save the data to your `scratch` directory.

If the MARS Retrieval icon specifies retrieval of fields, the resulting file is a GRIB file, which by default is assigned the icon name with extension `.grib`. If the icon specifies retrieval of observation data, the resulting file is a BUFR file, which by default is assigned the icon name with extension `.bufr`.

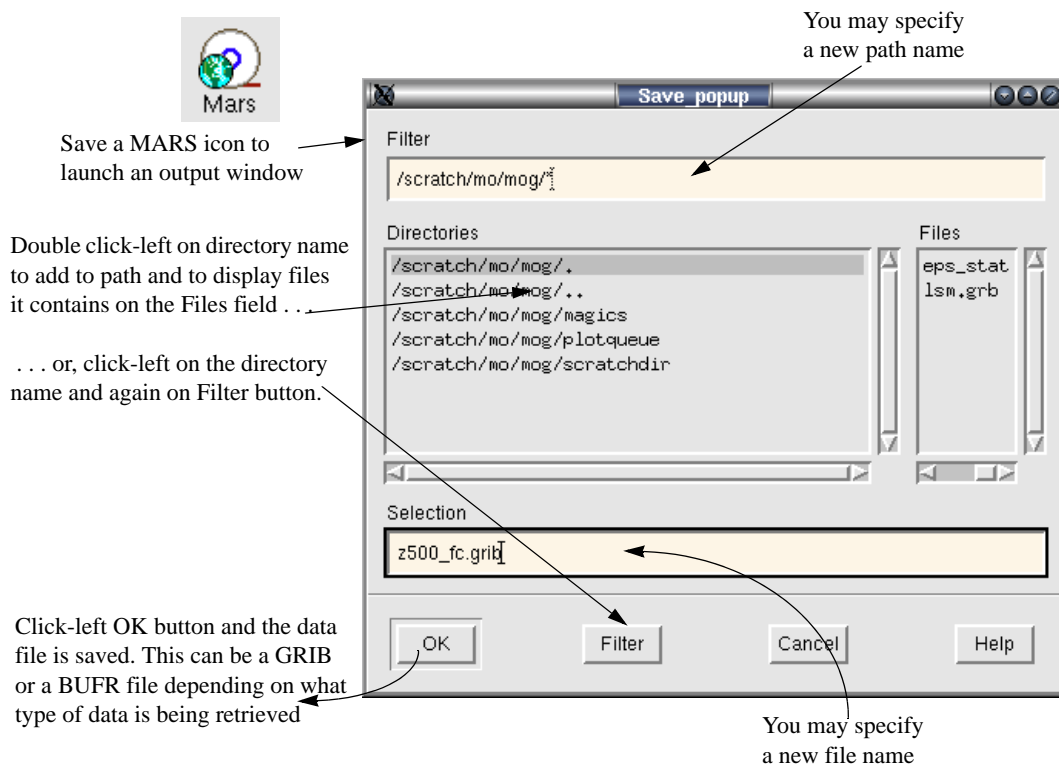


Figure III-1 Using an output window from a Save action on a MARS Retrieval icon to save retrieved data as a GRIB file.

Examining GRIB Data

To analyse the structure of your request and the contents of its corresponding GRIB header (plus a sample of its first 20 data values), use the **Examine** option from the icon menu. If the data has not yet been retrieved (i.e. it is not in the cache) the *Examine* action also carries out the retrieval and places the data in the cache. An Examine window (see Figure III-2) is launched which you use to inspect the data.

Note that you get exactly the same result if you were to *Examine* the GRIB file icon you can create from the request.


Figure III-2 presents an example : 96 fields are retrieved - forecasts for 2 parameters (Z and T) at 2 levels (1000 and 850 hPa) made on 3 different dates, verifying at 4 time steps later at 2 different hours (00 and 24). Examine windows may contain a number of levels represented by folders. Note that the structuring of the folders reflects the data organisation within MARS - in this example the folder sequence is : date - hour - time step - data (2 parameters at two levels), in accordance with the concept that each MARS file contains one date / one time / one forecast step at all levels for all parameters.

The final folder window shows icons for the data fields. The data field icon picture is different for different data representations :

Note that the sequence of fields as displayed in this last folder of the examine window *is usually not the same* as the internal sequence of fields in the GRIB file. The fields are ordered alphabetically so that the parameter with the lowest MARS code number appears first, and fields of the same parameter appear in ascending order of pressure or model level. The actual order of the fields in the retrieval is the order in which they were specified

The individual field icons can be dropped inside a plot window pane to be visualised, or dropped on a workspace (open folder or main UI). In this latter case, you get a new GRIB file icon on the workspace named Field N of GRIB_file_name. The N refers to the actual order number of that field inside the (source) GRIB file.

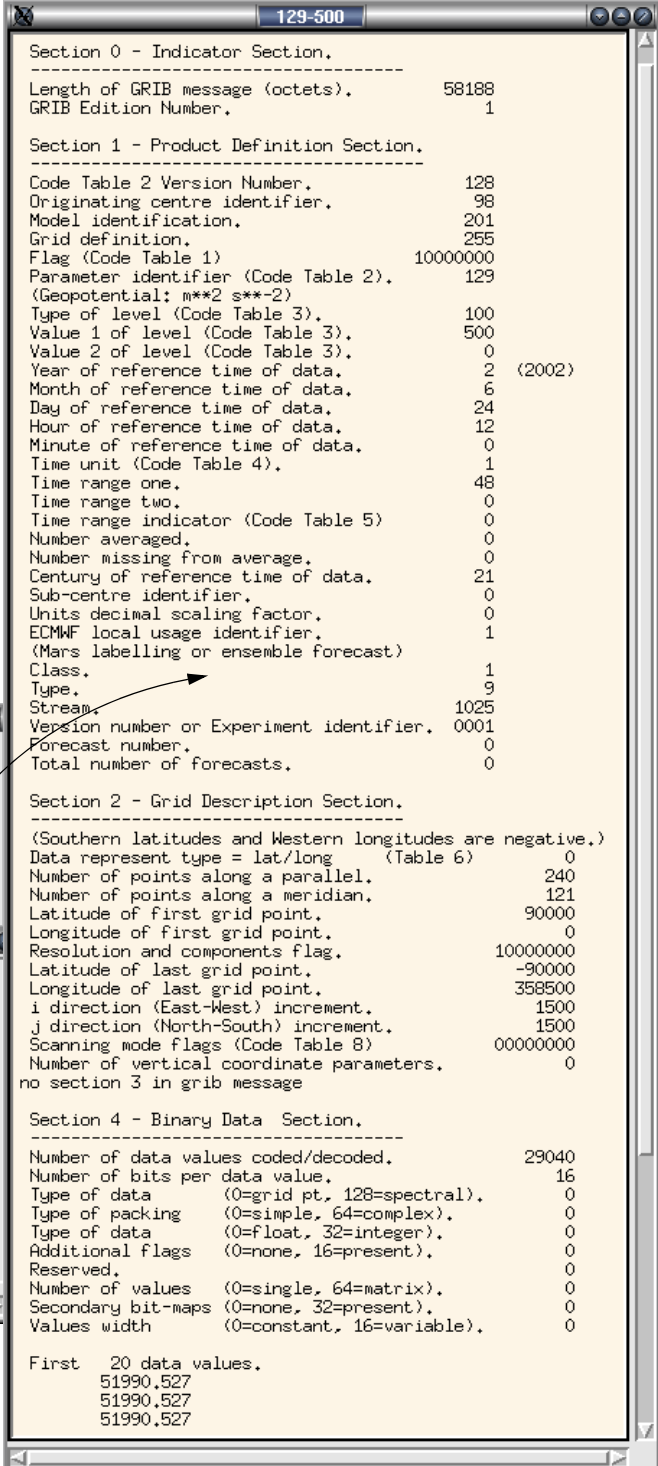
1 - Examine a MARS icon to launch an examine window



2 - Further windows with folder icons may appear, depending on what you retrieve. Double-click on a folder icon to open it. Repeat as needed

3 - The final stage : a window with icons for each data unit. Double-click the icon to reveal the GRIB file header window.

The GRIB file header window : Relevant sections of the GRIB header contents are listed in this window, as well as the first 20 values of data.



```

Section 0 - Indicator Section.
-----
Length of GRIB message (octets).      58188
GRIB Edition Number.                  1

Section 1 - Product Definition Section.
-----
Code Table 2 Version Number.          128
Originating centre identifier.         98
Model identification.                  201
Grid definition.                       255
Flag (Code Table 1)                   10000000
Parameter identifier (Code Table 2).   129
(Geopotential; m**2 s**-2)
Type of level (Code Table 3).          100
Value 1 of level (Code Table 3).       500
Value 2 of level (Code Table 3).       0
Year of reference time of data.        2 (2002)
Month of reference time of data.       6
Day of reference time of data.         24
Hour of reference time of data.        12
Minute of reference time of data.      0
Time unit (Code Table 4).              1
Time range one.                        48
Time range two.                        0
Time range indicator (Code Table 5)    0
Number averaged.                      0
Number missing from average.           0
Century of reference time of data.     21
Sub-centre identifier.                 0
Units decimal scaling factor.          0
ECMWF local usage identifier.          1
(Mars labelling or ensemble forecast)
Class.                                  1
Type.                                   9
Stream.                                 1025
Version number or Experiment identifier. 0001
Forecast number.                       0
Total number of forecasts.             0

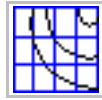
Section 2 - Grid Description Section.
-----
(Southern latitudes and Western longitudes are negative.)
Data represent type = lat/long (Table 6) 0
Number of points along a parallel.       240
Number of points along a meridian.      121
Latitude of first grid point.           900000
Longitude of first grid point.          0
Resolution and components flag.         10000000
Latitude of last grid point.            -90000
Longitude of last grid point.           358500
i direction (East-West) increment.     1500
j direction (North-South) increment.    1500
Scanning mode flags (Code Table 8)     00000000
Number of vertical coordinate parameters. 0
no section 3 in grib message

Section 4 - Binary Data Section.
-----
Number of data values coded/decoded.    29040
Number of bits per data value.          16
Type of data (0=grid pt, 128=spectral). 0
Type of packing (0=simple, 64=complex). 0
Type of data (0=float, 32=integer).     0
Additional flags (0=none, 16=present).  0
Reserved.                               0
Number of values (0=single, 64=matrix). 0
Secondary bit-maps (0=none, 32=present). 0
Values width (0=constant, 16=variable). 0

First 20 data values.
51990,527
51990,527
51990,527
    
```

Figure III-2 Examine action on a MARS Retrieval. It launches an examine window. In this example 96 fields are retrieved - forecasts for 1 parameter (Z) at 6levels (300, 400, 500, 700, 850 and 1000 hPa) made on 2 different dates, verifying at 5 time steps. The structuring of the folders reflects the data organisation within MARS (see text for details).

LATLONG MATRIX



This icon is a data icon for files of regular gridded lat-long data in ASCII format (ASCII matrix). Being a data icon, it doesn't exist "on its own", you create it by creating/modifying a data file according to some predetermined format that Metview understands. Once you do this a LatLong Matrix icon is automatically created by Metview.

From ASCII Data File to LLMatrix Icon

An ASCII file containing a regular grid of Lat Long data with individual values separated by blank spaces (spaces, tabs) and rows separated by newline characters, is known as an LLMatrix (Latitude Longitude Matrix).

With a simple addition to the file you can circumvent the need to encode the data in GRIB format - Metview assigns a particular icon to this type of ASCII data files and they can be used just like GRIB files. The procedure is as follows :

- Edit the file *outside* the Metview environment - if the file is brought into the Metview environment prior to being modified as described below, it is assigned a Notes icon which can't be modified afterwards. If you have done this, delete the Notes icon (you may need to copy the file back to a storage folder outside the Metview environment, using the UNIX prompt).
- Add some lines of information to the beginning of the file, then save it - these header lines provide required information - grid mesh, geographical extent, parameter name, ..., that allow Metview to "understand" the file.
- Move/copy/link the file into the Metview environment. When this is done, Metview assigns an LLMatrix icon to the file automatically as it recognises the format details of the header you have introduced in the previous step.

The file is now ready to be used as a normal GRIB file (as input, visualisation, etc). You can also convert it to a GRIB file through the icon Save option (see "Actions on the LLMatrix Icon" below).

There is no macro language equivalent - use macro `read()` function to read the ASCII data (see "ASCII matrix input" on page II- 33 and "General ASCII input" on page II- 34).

An alternative to this procedure is to use the Matrix icon which does not require you to introduce a header in the ASCII file and allows you to specify a path for a remote file. It is however more limited in scope.

Adding a Header to an ASCII Data File

The modification required to make Metview understand an ASCII Matrix data file, consists of adding a few lines to the beginning of the ASCII data file :

- The first line must contain the keyword `#LLMATRIX`

- Then follow with a variable number of elements (one per line) such as grid mesh, geographical extent, parameter name, ...
- Then add a line with the keyword #DATA

The original data follows these lines. Schematically your file will look like :

```
#LLMATRIX
...LLMatrixHeader components here...
#DATA
253 256 252 254 ...
255 253 253 256 ...
...
```

The LLMatrixHeader elements that you have to add (the lines between #LLMATRIX and #DATA) are detailed (list and description) in "LLMatrix Header Components" below. The #LLMATRIX and #DATA lines act as delimiters, separating data from header elements.

LLMatrix Header Components

The header parameters which you can specify are as follows :

NORTH

Geographical parameter. Enter the northernmost coordinate of your field (southern latitudes are negative), e.g. NORTH = 70. This parameter must always be given

WEST

Geographical parameter. Enter the westernmost coordinate of your field (western longitudes are negative), e.g. WEST = -120. This parameter must always be given

NLAT

Geographical parameter. Enter the number of grid points from North to South, e.g. NLAT = 240. This parameter must always be given

NLON

Geographical parameter. Enter the number of grid points from West to East, e.g. NLON = 121. This parameter must always be given

GRID

Geographical parameter. Enter the grid resolution(s) in degrees as two numbers separated by a slash (E-W resolution and N-S resolution), e.g. GRID = 2.5/2.5. This parameter must always be given

HSCAN

Ordering parameter. Enter the data horizontal scanning direction as a two letter string : WE for West to East (default), or EW for East to West. This parameter is optional.

VSCAN

Ordering parameter. Enter the data vertical scanning direction as a two letter string : NS for North to South (default), or SN for South to North. This parameter is optional.

SCANDIR

Ordering parameter. Enter the data primary scanning direction as a string : HORIZONTAL (default) to read data row by row or VERTICAL to read data column by column. Depending on which you choose you may have to specify either HSCAN or VSCAN. This parameter is optional.

MISSING

Ordering parameter. Enter the value used for missing values within your data matrix, e.g. MISSING = -99. The default value is the maximum floating point value of the machine you are using (hence unlikely to occur in a data file). This parameter is optional.

PARAM

Contents parameter. Enter the value of the parameter number, e.g. PARAM=129. The default value is 255, a GRIB generic value. These numbers are defined in GRIB Table 2 (e.g. Z = 129, T = 130). There are several versions of this table and you may use parameter TABLE2 to specify which version you want to use. This parameter is optional.

TABLE2

Contents parameter. Enter the version number of GRIB Table 2 that contains the definition of the parameter PARAM., e.g. TABLE2=128. Version numbers 128-254 are reserved for local use, while other numbers are defined by WMO for international exchange (currently using 1). The default value is 128, for the default ECMWF GRIB Table 2. Other local tables have been defined at ECMWF, such as 131, 140, 151, 160, 170, 254. This parameter is optional.

LEVEL

Contents parameter. Enter the pressure level of data in hPa, e.g. LEVEL = 500. Model levels are not catered for. If LEVEL is not given, surface level is assumed. This parameter is optional.

CENTRE

Contents parameter. Enter the code of the generating centre, e.g. CENTRE=98 (default). Use your own centre code if outside ECMWF (see GRIB Table 0). This parameter is optional.

DATE

Time parameter. Enter the date of your data as a string with the format YYYYMMDD.hh, e.g. DATE=19980425.25 (= 06h00 of the 25th April 1998). Note how the time of day is expressed as a fraction of the date. Alternatively, you can use dates relative to today, e.g. DATE=-2 (two days ago), but these cannot handle time of the day. Default date is 0, i.e. today. This parameter is optional.

FCAST

Time parameter. Forecast time step in hours, e.g. FCAST=48. Use this if your data is forecast data valid at DATE+FCAST. If FCAST is not used or it is set to 0, the data is interpreted as being analysis.

The example below shows the header of a LLMatrix with data starting at 90N, 0W covering the whole globe with a grid of mesh 1.5 degrees; hence dimensions are $240=1+(358.5-0)/1.5$ by $121=1+(90-(-90))/1.5$. Data is organised by row and is to be scanned from West to East. Data is analysis of 850mb Temperature for 06h00 of the 4th January 1998. The LLMatrixHeader has the following format :

```
#LLMATRIX
NORTH = 90
WEST = 0
GRID = 1.5/1.5
NLAT = 121
```

```
NLON = 240
HSCAN = WE
SCANDIR = H
PARAM = 130
LEVEL = 850
DATE = 19980104.25
#DATA
251 253 252 252 ...
253 253 251 251 ...
...
```

Actions on the LLMatrix Icon

This icon can be visualised by carrying out a visualise action or dropping it into a plot window. The actions available for the LLMatrix icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate*, *Delete* and *Output*. Those requiring some information are :

Execute - Reads the data and place it in a cache. No visualisation takes place

Visualise - visualises the data, using the default display window. For details on data visualisation see "Visualisation - an Overview" on page I- 62.

Examine - examines the contents of the file in the same way as for a GRIB icon : see "Examining GRIB Data" on page III- 31 . GRIB header information is from the LLMatrixHeader.

Save - this action allows you to convert the ASCII data into a GRIB file. It launches a save/browse window where you specify where to save the GRIB file and its name (icon name with extension .grib by default). It works in the same way as for a MARS Retrieval icon : see "Saving MARS Retrieval Output (as Files)" on page III- 30

MATRIX



This icon enables Metview to handle regular Lat-Long matrix data, i.e. data stored in an ASCII file containing a regular grid of Lat Long data with individual values separated by blank spaces (spaces, tabs) and rows separated by newline characters.

You need not use this icon - you can make Metview handle this type of data files by adding an header to the beginning of the file - see full details in "LatLong Matrix" on page III- 33. There are differences between using one or the other:

- using the Matrix icon you do not need to modify the original file
- you can handle the file remotely without having to link to it
- you can handle data arranged in Gaussian grids
- it can't handle as many different data layouts as the "add a header" approach of the Lat-Long Matrix

The Matrix Editor

Path

Specifies the full file name (including pathname). It is your responsibility to get the file type correctly and to make sure you have the right permissions to access it - Metview is unable to check this for you.

Field Organisation

The options are **Regular**, **Fitted** or **Gaussian**. Use **Regular** (other options not in use). For irregularly spaced point data, you must use the GeopointstoGRIB icon - see "Geopoints To GRIB" on page III-59.

Initial Point

Specifies the Latitude/Longitude coordinates of the first data point of your file.

Longitude Step

Specifies the longitude grid mesh in degrees. If your data goes East to West, the step must be negative.

Latitude Step

Specifies the latitude grid mesh in degrees. If your data goes North to South, the step must be negative.

Primary Index

Specifies the coordinate that changes faster - **Longitude** if your data points are organised as a sequence of lines of equal latitude and **Latitude** if as a sequence of lines of equal longitude.

Dimension

Specifies the number of x data points or number of columns, and the number of y data points or number of rows, as X/Y.

Actions on the Matrix icon

In addition to the four base actions (*Edit, Duplicate, Delete*) you have the action *Visualise* available :

Visualise - this action carries out the visualisation of the data, using the default display window. For details on data visualisation see "Visualisation - an Overview" on page I- 62.

ECFS (ECMWF ONLY)



This icon retrieves your files from ECFS - European Centre File Store. Note that this application fetches the data into the Metview cache. *The file is not saved* (i.e. it will be removed when you quit Metview), unless you apply the **Save** action in the icon menu

The macro language equivalent is `ecfs()`

The ECFS Editor

The editor for this icon has only two input parameters - the ECFS domain and the name of the file you want to retrieve from ECFS

Ecfs Domain

Specifies the ECFS domain. By default this is `ec:` and so far it is the only valid input.

File Name

Specifies the name of the file to be retrieved from ECFS. The name specified must not include the `ec:` prefix, but should include the ECFS path, e.g. `/uid/dir1/.../filename`. If the file resides in the root ECFS directory, you can only specify the file name.

Detailed information on ECFS can be obtained from the UNIX prompt. Given that ECFS commands mimic corresponding UNIX commands, you will obtain a man page for ECFS commands by entering :

```
% man ecfs
```

Actions on the ECFS icon

The actions available for the ECFS icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate*, *Delete* and *Output*.

Execute - this action retrieves the specified file from ECFS. The data is put in a cache, so that subsequent actions happen much faster.

Visualise - this action visualises the file retrieved from ECFS. If not visualised / executed before it also carries out the retrieval. The data is left in a cache, so that subsequent actions happen much faster. For details on visualisation see "Visualisation - an Overview" on page I- 62.

Examine - this action allows you to examine the contents of the retrieved file in the same way as you would a GRIB icon : see "Examining GRIB Data" on page III- 31.

Save - this action allows you to save the retrieved field as a GRIB file. It works in the same way as for a MARS Retrieval icon : see "Saving MARS Retrieval Output (as Files)" on page III- 30.

ODB DATABASE (ECMWF ONLY)



This icon specifies an ODB database. It contains only the database path using the syntax defined by the ODB API.

The only Icon Menu action assigned to this icon is *Examine*, which will start up the Metview ODB Browser to investigate the meta data (tables, columns, variables etc.) of the database.

The macro equivalent is `odb_database()`.

The ODB Database Editor

Database

Specifies an ODB database for that the query will be performed via the ODB API. For local databases the direct access mode is used providing a fast data retrieval. Databases on remote hosts can be communicated with in the remote access mode using the ODB client/server interface. In the remote access mode Metview acts as a client sending the ODB/SQL query request to the ODB server that performs the retrieval and passes the data back to Metview. Please note that this mode requires the ODB server running on the remote host. However, the advanced version of the ODB client/server interface that Metview is now based on automatically starts up the ODB server when required.

The database can be specified in various ways:

- `datapath`

This specification can be used only if the database type can be guessed from the datapath. For example, if the database is specified by the path `"/hugetmp/data/ECMA/ECMA.conv"` it will be identified as type of ECMA. In this case the client/server communication is bypassed and the direct access is used instead.

- `odb://host/datapath/DBTYPE`
- `odb://host:port/datapath/DBTYPE`

These specifications will automatically invoke the client/server communication even for local databases. Please note the the database type (DBTYPE) now needs to be explicitly specified!

- `host:/datapath`

This specification can be used only if the database type can be guessed from the datapath. It will automatically invoke the client/server communication even for local databases.

It is allowed to use wildcard format in the database specification. Users should basically be able to use any UNIX regular expression. For example if there are two databases (e.g. `ECMA.conv` and `ECMA.satob`) in the same directory (e.g. in directory `"datapath"`) the following is a valid database specification:

```
datapath/ECMA.{conv,satob}
```

and ODB API will handle the two databases as if they were a single one.

Another example shows how to specify all the databases whose name match the pattern ECMA.*:

```
datapath/ECMA.*
```

However, there are limitations of the wildcard notation. It is not possible to mix database types (e.g. CCMA with ECMA). The hostname cannot contain wildcard characters, either

Actions on the ODB Database icon

The only action available for the ODB Database icon is *Examine*.

Examine - The only Icon Menu action assigned to this icon is *Examine*, which will start up the Metview ODB Browser to investigate the meta data (tables, columns, variables etc.) of the database.

The ODB Browser

The *Examine* Icon Menu action assigned to the ODB Database icon will start up the Metview ODB Browser to investigate the meta data (tables, columns, variables etc.) of the database.

The ODB Browser retrieves the meta data information from the database via the ODB API. Its graphical interface is divided into four tabs:

- **General information:** contains a list with general information about the database such as database path, database type, number of pools, number of tables etc.
- **Table tree:** This tab is divided into two parts. The upper part displays the ODB table hierarchy. Each table in the hierarchy can be selected by clicking on it. When a table is selected its colour turns to red and all the columns contained by the table are displayed in the list in the lower part of the interface.
- **Columns:** This tab contains a list of all the columns in the database.
- **Variables:** This tab contains a list with all the predefined (SET) ODB variables.

ODB ACCESS (ECMWF ONLY)



This icon performs a user-defined ODB retrieval via the ODB API. For local databases the direct ODB access mode is used providing a fast data retrieval. Databases on remote hosts can be communicated with in the remote access mode using the ODB client/server interface. In the remote access mode Metview acts as a client sending the ODB/SQL query request to the ODB server that performs the retrieval and passes the data back to Metview. Please note that it requires the ODB server running on the remote host. However, the advanced version of the ODB client/server interface that Metview is now based on automatically starts up the ODB server when required.

Users can choose between retrieving the data either as a NetCDF or a geopoints file. Metview supports all the four geopoints file formats in ODB retrieval. For geopoints a wide range of data processing and visualisation operations are available (see "Geopoints" on page II- 23 for more details of the geopoints format and see also page II-172 for macro functions which extract ODB-specific meta-information from a geopoints file). However, the visualisation of NetCDF files will be available only in the forthcoming Metview 4 releases.

Providing details about ODB is beyond the scope of this User Guide. To use the ODB Access icon properly, especially to find information about how to define the ODB/SQL query, please study the ODB pages available online at the intranet page of the Satellite Section:

<http://intra.ecmwf.int/publications/cms/get/satds>

The macro language equivalent is `odb()`.

Metview Macro issues

The Query input element of the ODB Access icon accepts multi-line text.

In the Metview Macro language multi-line text can only be defined as inline string. Thus, when dropping an ODB Access icon into the Macro Editor the multi-line query text is automatically converted into an inline string. If the query contains only one line then the text remains the same in the macro and no inline string is involved.

The ODB Access Editor

Mode

Specifies the output format of the retrieval. The different options are as follows:

NETCDF: Unlike the geopoints output, for the NetCDF output the query is not restricted at all and users can specify all kinds of queries supported by the current ODB version. The result of the retrieval is a NetCDF file that is directly generated by the ODB API. The structure of the NetCDF file follows the ODB naming conventions. Having finished the query users can investigate the contents of the output file with the Metview NetCDF Browser by simply clicking on *Examine* in the icon menu.

GEO_STANDARD: The result is a "standard" format geopoints file that contains six columns: Latitude, Longitude, Level, Date, Time and Value. Please note that the production of this output format requires the careful setting of the **Query**, **Latitude**, **Longitude** and **Value** input elements (see below).

GEO_XYV: The result is a "XYV" format geopoints file that contains three columns: Longitude, Latitude and Value. Please note that the production of this output format requires the careful setting of the **Query**, **Latitude**, **Longitude** and **Value** input elements (see below).

GEO_XY_VECTOR: The result is a "XY_VECTOR" format geopoints file that contains seven columns: Latitude, Longitude, Level, Date, Time, X-Component and Y-Component. Please note that the production of this output format requires the careful setting of the **Query**, **Latitude**, **Longitude**, **Value** and **Value2** input elements (see below).

GEO_POLAR_VECTOR: The result is a "POLAR_VECTOR" format geopoints file that contains seven columns: Latitude, Longitude, Level, Date, Time, Speed and Direction. Please note that the production of this output format requires the careful setting of the **Query**, **Latitude**, **Longitude**, **Value** and **Value2** input elements (see below).

The default value is **GEO_XYV**.

Database

Specifies an ODB database for the query that will be performed via the ODB API. For local databases the direct access mode is used, providing a fast data retrieval. Databases on remote hosts are communicated with in the remote access mode using the ODB client/server interface. In the remote access mode Metview acts as a client sending the ODB/SQL query request to the ODB server that performs the retrieval and passes the data back to Metview. Please note that this mode requires the ODB server running on the remote host. However, the advanced version of the ODB client/server interface that Metview is now based on automatically starts up the ODB server when required.

The database can be specified in various ways:

- datapath

This specification can be used only if the database type can be guessed from the datapath. For example, if the database is specified by the path `"/hugetmp/data/ECMA/ECMA.conv"` it will be identified as type of ECMA. In this case the client/server communication is bypassed and the direct access is used instead.

- odb://host/datapath/DBTYPE
- odb://host:port/datapath/DBTYPE

These specifications will automatically invoke the client/server communication even for local databases. Please note the the database type (DBTYPE) now needs to be explicitly specified!

- host:/datapath

This specification can be used only if the database type can be guessed from the datapath. It will automatically invoke the client/server communication even for local databases.

It is allowed to use wildcard format in the database specification. Users should basically be able to use any UNIX regular expression. For example if there are two databases (e.g. `ECMA.conv` and `ECMA.satob`) in the same directory (e.g. in directory "datapath") the following is a valid database specification:

```
datapath/ECMA.{conv,satob}
```

and ODB API will handle the two databases as if it was a single one.

Another example shows how to specify all the databases whose name match the pattern `ECMA.*`:

```
datapath/ECMA.*
```

However, there are limitations of the wildcard notation. It is not possible to mix database type (e.g. CCMA with ECMA). Also the hostname cannot contain wildcard characters.

Database Icon

Specifies an ODB database icon (see "ODB Database (ECMWF only)" on page III- 41) defining the database for that the query will be performed. Please note that if an ODB database icon is present here the database specified in the **Database** input element will be ignored.

Query

Specifies the ODB/SQL query to retrieve data from the ODB database. The query must have the following syntax (for detailed description please turn to the ODB documentation):

```
[set variables]
select columns
from tables
[query conditions]
```

where:

`set variables` are optional user-defined ODB SET-variables, `columns` is a comma separated list of the odb columns to be retrieved. Aliases and functions are allowed to be used here. `tables` is a comma separated list of the odb tables that contain the specified ODB columns. `query conditions` are optional query conditions.

If the output format of the retrieval is NetCDF then there are no restrictions for the query and it can contain all the advanced features of the ODB/SQL query language.

However, for the geopoints output formats there are certain restrictions. Since geopoints is a fixed format, a mapping between ODB column names and geopoints variables must be ensured. This can be specified by setting the **Latitude**, **Longitude**, **Value** etc. icon input fields properly. For example, the *Latitude* icon input field specifies which ODB column is interpreted as the latitude variable of the resulting geopoints file. However, please note that by using the columns in the proper order in the query and using the default settings for the icon input fields the explicit specification of the mapping can be skipped (see below).

The geopoints format requires the latitude and longitude data to be in degrees. Since the latitude and longitude units cannot be determined automatically from ODB Metview does not convert this data to degrees. Instead, users have to ensure that the retrieved latitude and longitude data are in degrees. This can be achieved e.g. by using the `ldegrees()` ODB/SQL query function.

For each geopoints format there can be at least three columns in the query: a latitude, a longitude and a data value column (specified in icon input fields **Latitude**, **Longitude** and **Value**, respectively). For the geopoints vector outputs the second data value column must be also presented (specified in the icon input field **Value2**). Columns corresponding to the **Level**, **Date** and **Time** icon input fields can be also specified optionally. Other columns may be added to the query but please note that these columns will not appear in the resulting geopoints file since it can store only one or two data value columns. Thus there is no point in adding these extra columns to the query.

We provide an example below for each geopoints format in order to illustrate the rules to be applied to the query and the corresponding input elements:

GEO_STANDARD

Example: Retrieval of the temperature first guess departures and pressure for Aircraft observations.

Query:

```
select lat, lon, fg_depar, press
from hdr, body
where obstype=2 and varno=2
```

Input elements:

If is nothing specified (default value) for **Latitude**, **Longitude** and **Value** the first three columns of the query will be interpreted automatically as Latitude, Longitude and Value. Thus the following setting of input elements is valid:

```
Latitude=
Longitude=
Level=press@body
Value=
```

Alternatively we can explicitly specify the mapping between column names and geopoints variables:

```
Latitude=lat@hdr
Longitude=lon@hdr
Level=press@body
Value=fg_depar@body
```

GEO_XYV

Example: Retrieval of the temperature first guess departures for Aircraft observations.

Query:

```
select lat, lon, fg_depar
from hdr, body
where obstype=2 and varno=2
```

Input elements:

If is nothing specified (default value) for **Latitude**, **Longitude** and **Value** the first three columns of the query will be interpreted automatically as Latitude, Longitude and Value. Thus the following setting of input elements is valid:

```
Latitude=
Longitude=
Value=
```

Alternatively we can explicitly specify the mapping between column names and geopoints variables:

```

Latitude=lat@hdr
Longitude=lon@hdr
Value=fg_depar@body

```

GEO_XY_VECTOR

Example: Retrieval of the U and V wind components for Aircraft observations.

Query:

```

select lat, lon, press, obsvalue, obsvalue#1
from hdr, body
where obstype=2 and varno=3 and varno#1=4

```

If is nothing specified (default value) for **Latitude**, **Longitude**, **Value** and **Value2**, the first four columns of the query will be interpreted automatically as Latitude, Longitude, Value and Value2. Thus the following setting of input elements is valid:

```

Latitude=
Longitude=
Value=
Value2=

```

Alternatively we can explicitly specify the mapping between column names and geopoints variables:

```

Latitude=lat@hdr
Longitude=lon@hdr
Value=obsvalue@body
Value2=obsvalue@body#1

```

GEO_POLAR_VECTOR

Example: Retrieval of the wind speed and direction for Aircraft observations.

Query:

```

select lat, lon, press, speed(obsvalue, obsvalue#1),
dir(obsvalue,
obsvalue#1)
from hdr, body
where obstype=2 and varno=3 and varno#1=4

```

If is nothing specified (default value) for **Latitude**, **Longitude**, **Value** and **Value2**, the first four columns of the query will be interpreted automatically as Latitude, Longitude, Value and Value2. Thus the following setting of input elements is valid:

```

Latitude=
Longitude=
Value=
Value2=

```

Alternatively we can explicitly specify the mapping between column names and geopoints variables:

```

Latitude=lat@hdr
Longitude=lon@hdr
Value=speed(obsvalue@body, obsvalue@body#1)
Value2=dir(obsvalue@body, obsvalue@body#1)

```

Packing

This input element is active only for the NetCDF output and specifies packed or unpacked output file generation. In an unpacked output file all the retrieved ODB data (except strings) are stored as double precision floating point variables. If packing is selected the non-string data of the ODB retrieval will be scaled in the following way:

$$\text{odb_data} = \text{scale_offset} + \text{scale_factor} * \text{odb_data_in_netcdf}$$

and the type of the NetCDF variable is selected according to the scaled data range. The "scale_offset" and "scale_factor" values are specified in the attributes of each NetCDF variable.

The default value is Yes.

Latitude

This input element is active only for the the geopoints output and specifies the ODB column name or alias in the query to be interpreted as the latitude of the observations. If nothing is specified here Metview will automatically interpret the first column in the query as latitude.

The default value is an empty string.

Longitude

This input element is active only for the the geopoints output and specifies the ODB column name or alias in the query to be interpreted as the longitude of the observations. If nothing is specified here Metview will automatically interpret the second column in the query as longitude.

The default value is an empty string.

Value

This input element is active only for the the geopoints output and specifies the ODB column name or alias in the query to be interpreted as the value data or the x vector component or the speed in the resulting geopoints file (depending on the format). If nothing is specified here Metview will automatically interpret the third column in the query as value, x vector component or speed, respectively.

The default value is an empty string.

Value2

This input element is active only for the the GEO_XY_VECTOR and GEO_POLAR_VECTOR output formats and specifies the ODB column name or alias in the query to be interpreted as the y vector component or the direction in the resulting geopoints file (depending on the format). If nothing is specified here Metview will automatically interpret the fourth column in the query as y vector component or direction, respectively.

The default value is an empty string.

Level

This input element is active only for the the GEO_STANDARD, GEO_XY_VECTOR and GEO_POLAR_VECTOR output formats and specifies the ODB column name or alias in the query to be interpreted as the level of the observations. This field is optional.

The default value is an empty string.

Date

This input element is active only for the the GEO_STANDARD, GEO_XY_VECTOR and GEO_POLAR_VECTOR output formats and specifies the ODB column name or alias in the query to be interpreted as the date of the observations. This field is optional.

The default value is an empty string.

Time

This input element is active only for the the GEO_STANDARD, GEO_XY_VECTOR and GEO_POLAR_VECTOR output formats and specifies the ODB column name or alias in the query to be interpreted as the time of the observations. This field is optional.

The default value is an empty string.

Number of Rows

Specifies the maximum number of rows in the ODB retrieval. This option exists to control the size of the retrieved ODB data since it can contain even millions of lines, resulting in unwanted performance issues.

The default value is 500000.

Actions on the ODB Access icon

The actions available for the ODB Access icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate*, *Delete* and *Output*.

Execute - this action retrieves the specified data from the ODB database. The data is put in a cache, so that subsequent actions happen much faster.

Visualise - this action visualises the file retrieved from the database. If not visualised / executed before, it also carries out the retrieval. The data is left in a cache, so that subsequent actions happen much faster. For details on visualisation see "Visualisation - an Overview" on page I- 62.

Examine - this action allows you to examine the contents of the retrieved data in NetCDF or geopoints format (depending on the output format).

Save - this action allows you to save the retrieved field as a NetCDF or geopoints file (depending on the output format). See "Geopoints" on page II- 23 for more details of the geopoints format.

GRIB FILTER



The GRIB Filter icon is used to filter GRIB data from either a MARS Retrieval icon or a GRIB file icon. You can then obtain just the field(s) you want from a large request/file containing a wide variety of other field(s). As an added feature it can also read (*but not filter!*) BUFR files placed *outside* your Metview directory (see "Observation Filter" on page III- 53, for filtering of BUFR data).

You may ask yourself, why filter a MARS Retrieval instead of retrieving from the database exactly the data you want, thus avoiding the extra filtering step? The usefulness depends on what you want to do :

It may be more efficient to retrieve (and save) a large chunk of data with a single MARS Retrieval and then filter fractions of the large dataset with GRIB Filter. Remember that the large chunk of data, once retrieved is cached (or you may have saved it) and further access to it via GRIB Filter is much faster than making many separate individual MARS Retrievals.

Note that this icon is currently named Data in File within the Metview User Interface and is informally referred to as the "read icon" or "diskette icon".

The macro language icon function is the `read()` function

The GRIB Filter Editor

With the exception of two parameters which specify the data to be filtered, all other parameters found in the GRIB Filter editor can also be found in the MARS Retrieval editor.

Source

Specifies the GRIB file path and name. Type the full file path and name.

Data

Drop a MARS Retrieval icon or a GRIB file icon inside this icon field. For help see "Icon fields" on page I- 40.

Order

Specifies whether you want to sort the files you have filtered (Sorted) or to obtain them in the order they are on the original file (As Is). The sorting is the same as that carried out by MARS on retrieving data (date, time, forecast step, level, parameter), hence this has no effect if you are filtering a MARS request (or a GRIB file containing the result of a single MARS retrieval).

Note that you should specify either **Source** or **Data**, *not both*, otherwise the process will fail. The remainder of the parameters are filtering keys : the icon outputs only the data which obeys *all* filter conditions. If a parameter is not to act as a filter key, leave set at `ANY`. Refer to "The MARS Retrieval Editor" on page III- 18, for details on the other listed parameters.

Actions on the GRIB Filter Icon

The actions available for the GRIB Filter icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate*, *Delete* and *Output*. Those requiring some information are :

Execute - this action carries out the retrieval/filtering of the data. If you supplied a MARS Retrieval icon as input data, the data is retrieved from the archives and placed in the cache, from where GRIB Filter extracts the data according to the specified keys. Using a GRIB file as Source, data is filtered from it. All icons involved are signalled as active during the process. Note that the data is not automatically visualised.

Visualise - this action carries out the visualisation of the data, using the default plot window. If the icon has not been executed or visualised before, the visualise action also carries out the retrieval/filtering of the data. For details on data visualisation see "Visualisation - an Overview" on page I- 62.

Examine - this action allows you to examine the contents of the filtered data. It works in the same way as for a MARS icon : see "Examining GRIB Data" on page III- 31.

Save - this action allows you to save the filtered data as a GRIB file. Again it works in the same way as for a MARS icon : see "Saving MARS Retrieval Output (as Files)" on page III- 30.

OBSERVATION FILTER



The Observation Filter icon filters observation data from a BUFR file or MARS Retrieval of observation data and produces output either as a BUFR file or as a geographical points (geopoints) file. You can use an Observation Filter as input to another Observation Filter as long as the first produces BUFR output.

BUFR stands for "Binary Universal Form for Data Representation", the format in which observation data (in the broadest sense including ground observations, radiosonde soundings, radar and satellite images and soundings) are stored. Full details can be found in the WMO references provided (see "Data Formats" in "Brief Overview of MARS" on page III- 13) and the ECMWF Meteorological Bulletin M1.4/4 "BUFR User Guide and Reference Manual" (1994).

Very (very) briefly, a BUFR file is composed of BUFR messages. Each message has header sections containing information (e.g. type, subtype, date and time) and a section containing one or more observations in a coded format. Each observation contains several values of different observed physical quantities (e.g. SYNOP, TEMP, PILOT, METAR,...). Observations are classified by Type (defined by WMO) and Subtype (which may be particular to each institution).

Geopoints files are ASCII files containing single or paired parameter (observed physical quantity) values together with space and time coordinates.

If you need observations of a given parameter for combination with other types of data, e.g. to derive differences between forecast fields and observations, they are required to be in geopoints format. You must use this icon to filter out the single parameter values out of the BUFR file or archived observation data and return/save them as a geopoints variable/file.

The macro language icon function is the `obsfilter()` function.

Filtering Efficiency

You may filter observations according to a wide variety of parameters or combinations thereof : you may filter on date and time, on location (meteorological station, WMO block, user defined area, proximity to user defined line) and range of values.

Regarding the structure of the input BUFR file, note that some of the filtering parameters such as observation type, subtype, date and time are located in the header part of the BUFR message, whilst others are located in the data part of the BUFR message itself.

This implies that the filtering of BUFR data according to parameters located in the header does not require decoding of the remaining information and thus is considerably (about 10 times) faster. Internally, filtering is always done first on the header parameters (if specified).

The Observation Filter Editor

Data

Drop any icon containing or returning BUFR data. This may be a MARS Retrieval (of observations) icon, a BUFR file icon or an Observation Filter icon (provided it outputs BUFR, *not* geopoints). The default icon is a MARS Retrieval for 4 day old 12 UTC synoptic observations for the whole globe. See "Icon fields" on page I- 40 and "Icon input drawers" on page I- 45 for details on input of icons.

Output

Specifies the output format you want. Specify one of the geopoints formats (**Geographical Points**, **Geographical Polar Vectors** or **Geographical X Y Vectors**) if you want to plot just one or two parameters or if you want to do calculations (including calculations with GRIB fields - combining geopoints with GRIB fields outputs geopoints). The parameters that follow - **Parameter**, **Level**, **First Level**, **Second Level** and **Occurrence Index** - are not available if you specify **BUFR** output, as BUFR output must be formed by whole messages (of a given type).

Parameter

Available only for **Output** set to one of the geopoints formats (see above). To specify a parameter enter its unique descriptor value (a numerical code). **Geographical Polar Vectors** and **Geographical X Y Vectors** require two descriptors, separated by a slash (/). The descriptor value is of the form **XXYYY**, where **XX** defines the class (e.g. 12 = Temperature class) and **YYY** the parameter within that class (e.g. 12004 = Dry bulb Temperature at 2m). These descriptor values are different from the ones of the GRIB format. If you do not know the descriptor value, click-left on the assist button to obtain a check list of parameters and associated descriptors. This list only contains the most common parameters. If the one you need is not on the list you have to look up its descriptor value in the "BUFR User Guide and Reference Manual" (ECMWF Meteorological Bulletin M1.4/4) - see "BUFR Table B", pages 99-108. The significance of code and flag values for non-quantitative parameters are given in the same reference, in "BUFR code table", pages 111-154.

Missing Data

Available only for **Output** set to one of the geopoints formats (see above). If set to **Ignore**, missing data is not included in the output file; this is the default behaviour. If set to **Include**, missing data will be output to the geopoints file, its value being set to that specified by **Missing Data Value**. Note that when the output format is one of the two geopoints vector formats, the test for missing data is only performed on the first parameter.

Missing Data Value

Available only for **Output** set to one of the geopoints formats and **Missing Data** set to **Include**. Any missing observations will be output as this value (default 0). It is wise, therefore, to ensure that this value is outwith the range of possible values for the requested parameter(s). Note that when the output format is one of the two geopoints vector formats, the test for missing data is only performed on the first parameter.

Level

Available only for **Output** set to one of the geopoints formats (see above). Specify one of **Surface**, **Single Level**, **Layer** and **Occurrence**. What you specify here must be consistent with the parameter you specified for filtering.

First Level

Available only for **Output** set to one of the geopoints formats (see above) and **Level** set to **Single Level** or **Layer**. Specify the level of the observation in hPa. If **Layer** was chosen for **Level**, the value

will specify the bottom level of the layer. The assist button gives you a check list of most common pressure levels in hPa.

Second Level

Available only for **Output** set to one of the geopoints formats (see above) and **Level** set to **Layer**. Specify the top level of the layer in hPa.

Occurrence Index

Only available if **Level** set to **Occurrence**. Specify the numerical index of a parameter that has several values within one observation (e.g. cloud amount on different levels or water temperature at different depths).

Observation Types

Specifies the numerical code or text string for the desired observation type. The assist button provides a partial list of available text strings and associated types. Observation types are standardised by WMO and are fixed from place to place. See the "BUFR User Guide and Reference Manual" (ECMWF Meteorological Bulletin M1.4/4) - "BUFR Table A", page 97 - for a complete list of numerical codes.

Observation Subtypes

Specifies the numerical code or text string for the desired observation subtype. The assist button provides a list of available numerical codes and associated subtypes. Note that institutions are free to define their own subtypes hence these are not an international standard.

Date

Specifies observation(s) date. It is not possible to specify a range of dates. If you are filtering a new MARS Retrieval, remember that archived observations are always a couple of days old - trying to retrieve yesterday's observations is likely to fail. Allowed formats for the date are as follows :

- Absolute as YYYYMMDD
- Relative as -n; n is the number of days before today (-1 = yesterday)
- To specify a number of different dates use : YYYYMMDD1/YYYYMMDD2/ . . .

Specifying a value for **Date** requires setting a value for **Time** (if both are set to ANY, changing **Date** will change **Time** from ANY to 12).

Time

Specifies time of the observation(s). Required format is HHMM. It is not possible to specify a range of times.

Resolution in Mins

Specifies a time window in minutes around the value chosen for **Time**.

WMO Blocks

Specifies a WMO block number. These identify a geographical region, e.g. 02 for Sweden and Finland, 16 for Italy and Greece.

WMO Stations

Specifies a list of WMO stations, using the five digit station identifier (the first two of which are the WMO block number).

Location Filter

Specifies one of **No Location Filter**, **Area**, **Cross-Section Line**. This allows you to filter observations contained within a geographical area or within a given proximity to a geographical line between two points.

Area

Specifies the coordinates of the area of interest. Enter coordinates (lat/long) of an area separated by a "/" (top left lat and long, bottom right lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42)

Line

Specifies the coordinates of a transect line. Enter coordinates (lat/long) of a line separated by a "/" (easternmost lat and long, westernmost lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42). This will filter all stations close enough to the line - how close is defined by **Delta in Km**.

Delta in Km

Specifies the width of the cross section line defined in **Line**

Custom Filter

This allows you to filter observations of a given parameter according to its value. You can select observations equal to a value (**Filter by Value**) or within/outside a given range of values (**Filter by Range/Filter by Exclude Range**). Note that naturally you must specify one observed parameter to be filtered in this way.

Custom Parameter

Specifies the descriptor value of the parameter you want to filter according to value. For further information, see the details in "Parameter" on page III- 54.

Custom Values

Specifies the desired numerical values for filtering by value. You may specify more than one value, separated by a forward slash (e.g. n1/n2). If you Filter by Value, observations of the selected parameter with the value *equal to* n1 *or* n2 are selected (you may specify more than two values). If you Filter by Range, observations of the selected parameter with the value *within* the n1 *to* n2 interval are selected. If you Filter by Exclude Range, observations of the selected parameter with the value *outside* the n1 *to* n2 interval are selected.

Actions on the Observation Filter Icon

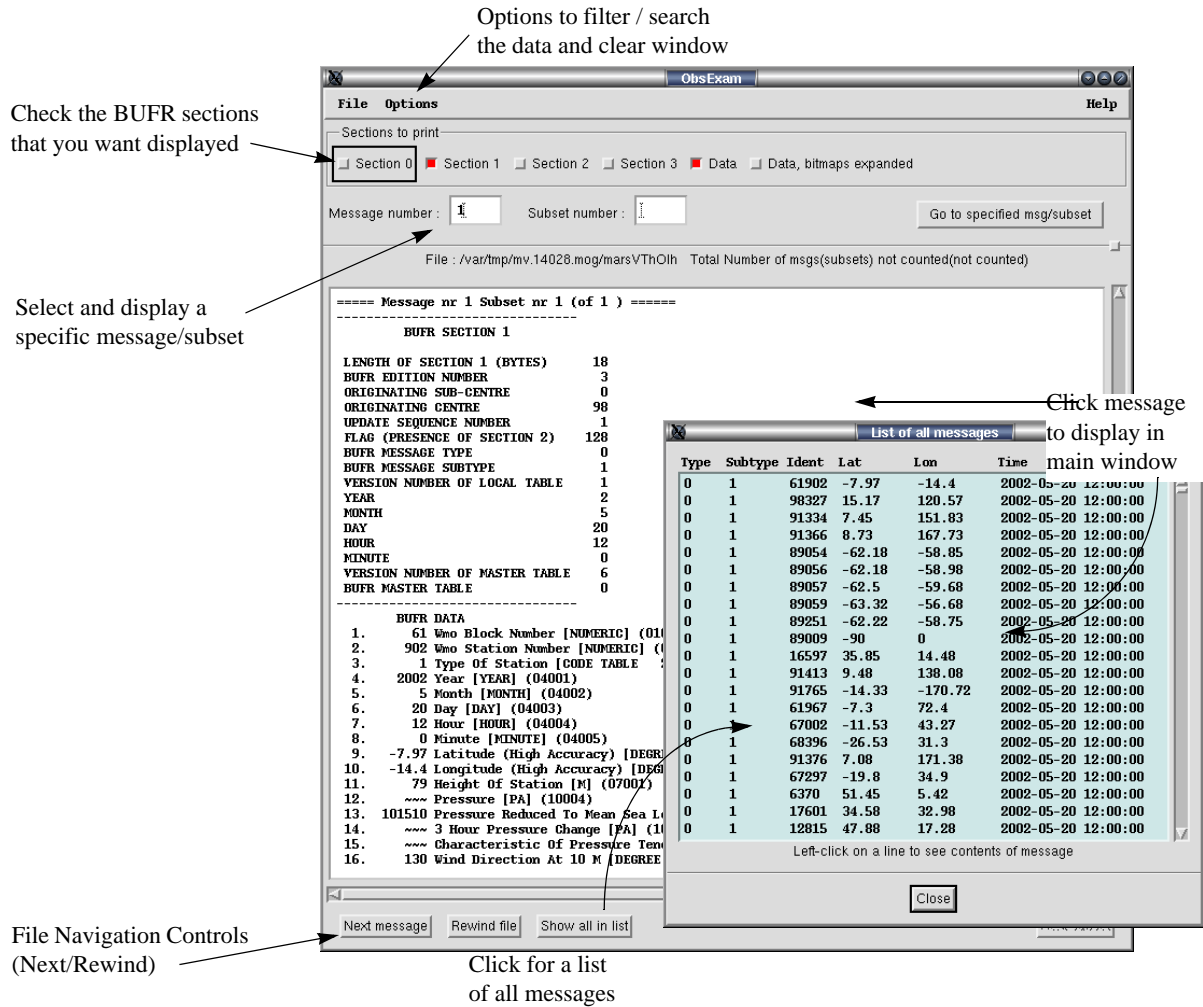
The actions available for the Observation Filter icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate*, *Delete* and *Output*. Those requiring some information are :

Execute - this action carries out the (retrieval and) filtering of the data which is then placed in the cache. All icons involved are signalled as active during the process. Note that the data is not automatically visualised.

Visualise - this action carries out the visualisation of the data, using the default display window. If the icon has not been executed or visualised before, the visualise action also carries out the retrieval/filtering of the data. For details on data visualisation see "Visualisation - an Overview" on page I- 62.

Save - this action allows you to save the filtered data as a BUFR or geopoints file. It works in the same way as for a MARS icon, in that an I/O window pops up where you specify the path and name for the destination file - this will be a BUFR or a geopoints icon with the same name of the icon and extension .bufr or .gpt - see "Saving MARS Retrieval Output (as Files)" on page III- 30.

Examine - this action provides information on the contents of the filtered data. Application OBSExam pops up displaying the contents of the BUFR file :



This application allows you to visualise any of the BUFR messages contained in the filtered BUFR data returned by the Observation Filter icon. It comprises navigation facilities and the capacity to filter subsets from the data and save them as new files.

GEOPOINTS TO GRIB



This icon interpolates irregularly spaced point data (in geopoints format) into a regular Lat-Lon matrix, which can then be plotted, saved or combined with other GRIB data. The interpolation method used is simple linear interpolation. Note that only the first parameter of a double-valued geopoints vector will be used.

The macro language equivalent is `geo_to_grib()`

The Geopoints To GRIB Editor

The editor for this icon is fairly simple. Briefly, you specify a geopoints icon for data input and define an area and a resolution for the interpolation.

Geopoints

Specifies the data to be interpolated. The data must be in geopoints format (e.g. a geopoints data icon or an Observation Filter icon returning geopoints - see "Observation Filter" on page III- 53). The icon input drawer should contain suitable icons for an example (default Observation Filter icon filtering into a geopoints format, 2m T data out of a MARS Retrieval of synoptic land observations). This can be easily modified to suit your requirements as long as the output is geopoints.

Area

Specifies a geographical area over which to carry out the interpolation, the default being for the whole globe. Enter coordinates (lat/long) of an area separated by a "/" (top left lat and long, bottom right lat and long); alternatively, use the assist button to define the area graphically (see "Geography help tool" on page I- 42).

Grid

Specifies a resolution in degrees, thus together with **Area**, determining the limits and density of the regular grid for interpolation of the point data values. Enter the longitude and latitude resolution as numbers separated by a "/".

Tolerance

Specifies a neighbourhood in degrees around each grid point. All geopoints data within this neighbourhood are used to interpolate the value at the central grid point. E.g. if **Tolerance** is 2 then all geopoints within a ± 2 degrees square around the grid point are used.

If your geopoints data has high spatial density then you can afford to specify a short neighbourhood, if the density is sparse you have to use a wide neighbourhood. Remember that the wider the neighbourhood the smoother the resulting interpolated field

Actions on the Geopoints To GRIB icon

The actions available for the Geopoints To GRIB icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate*, *Delete* and *Output*. Those requiring some information are :

Execute - this action processes the input data (reads a geopoints file or retrieves and filters observation data into a geopoints) and carries out the interpolation into the regular grid. The data is put in a cache, so that subsequent visualisations happen much faster.

Visualise - this action visualises the field derived from the interpolation procedure. If not visualised / executed before also carries out the data processing and interpolation of the data. The derived data is left in a cache, so that subsequent visualisations happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

Examine - this action allows you to examine the contents of the interpolated field. It works in the same way as for a MARS Retrieval icon : see "Examining GRIB Data" on page III- 31.

Save - this action allows you to save the interpolated data as a GRIB file. Again it works in the same way as for a MARS Retrieval icon : see "Saving MARS Retrieval Output (as Files)" on page III- 30.

GRIB To GEOPPOINTS



This icon takes data in GRIB format and converts it to geopoints format. This provides a quick and easy way to convert GRIB data into ASCII format.

The macro language equivalent is `grib_to_geo()`.

The GRIB To Geopoints Editor

Data

Specifies the GRIB data to be converted. Drag a suitable icon into the editor.

Geopoints Format

Specifies the output format. **Traditional** format stores Latitude, Longitude, Height, Date, Time and Value. **X Y V** Format stores Longitude, Latitude and Value. The difference can easily be seen by saving the data to a file and examining it with a text editor (via the *Examine* action).

Actions on the GRIB To Geopoints icon

The actions available for the GRIB To Geopoints icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate*, *Delete* and *Output*. Those requiring some information are :

Execute - this action processes the input data (reads a GRIB file or retrieves and filters data into a GRIB file) and carries out the conversion to a set of geopoints. The data is put in a cache, so that subsequent visualisations happen more quickly.

Visualise - this action visualises the geopoints derived from the conversion. If not visualised / executed before also carries out the data processing and interpolation of the data. The derived data is left in a cache, so that subsequent visualisations happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

Examine - this action allows you to examine the contents of the data. It brings the data into an ASCII text editor for examination.

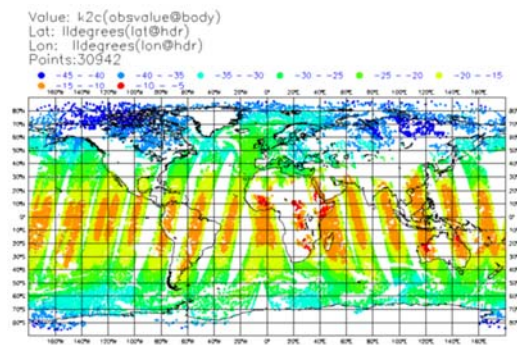
Save - this action allows you to save the geopoints data as an ASCII file.

GEOTOOLS

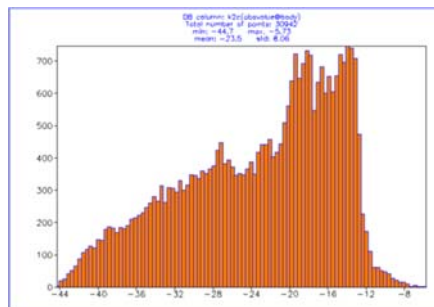


This icon was designed for the purpose of examining observation data in geoints format. It is intended as a temporary icon which will be replaced in Metview 4 by something more permanent. It can perform three functions:

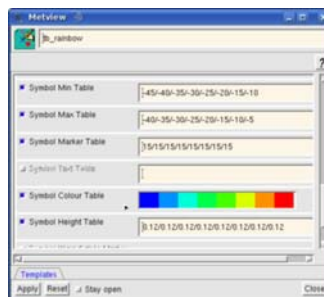
1) plot a preview of geoints data using an automatically-generated colour scale with an informative title and legend



2) plot a histogram of the data



3) generate a Symbol Plotting icon with an automatically calculated colour scale



The macro language equivalent is `geo_tools()`.

The GeoTools Editor

Mode

Specifies which of the above modes the icon should operate in.

Symbol Icon Name

Specifies the name of the Symbol Plotting icon to be generated if **Mode** is set to **Save Symbol Icon**. This path is relative to the directory containing the GeoTools icon.

Data

Specifies the data to be used. The data must be in geopoints format (e.g. a geopoints data icon or an Observation Filter icon returning geopoints - see "Observation Filter" on page III- 53; other possibilities are a Macro icon or an ODB Access icon).

Level Selection Type

Relevant only when **Mode** is set to **Preview**. Set to **Count** in order to specify the number of value levels (bands) (see **Level Count**). Set to **Interval** to specify the size of intervals (see **Level Interval**). Set to **List** to specify a 'hand-written' list of levels (see **Level List**).

Min Level

Relevant only when **Mode** is set to **Preview**. Data values below this value will not be plotted.

Max Level

Relevant only when **Mode** is set to **Preview**. Data values above this value will not be plotted.

Level Count

Relevant only when **Mode** is set to **Preview** and **Level Selection Type** is set to **Count**. Specifies the number of levels used, evenly spaced between the minimum and maximum data values.

Level Interval

Relevant only when **Mode** is set to **Preview** and **Level Selection Type** is set to **Interval**. Specifies the size of data intervals to be used.

Level List

Relevant only when **Mode** is set to **Preview** and **Level Selection Type** is set to **Level List**. Specifies the list of data intervals to be used.

Colour Method

Relevant only when **Mode** is set to **Preview**. A colour scheme is automatically generated using one of these methods. **Hue Cw**: interpolated clockwise through the HSL colour wheel; **Linear**: interpolated through RGB space; **Hue Ccw**: interpolated anti-clockwise through the HSL colour wheel.

Colour Min Level

Relevant only when **Mode** is set to **Preview**. The colours assigned to the data intervals are interpolated from **Colour Min Level** to **Colour Max Level**. Either choose a pre-set colour, or choose **Custom Rgb** and set **Colour Min Rgb**.

Colour Min Rgb

Relevant only when **Mode** is set to **Preview** and **Colour Min Level** is set to **Custom Rgb**. The colour is specified as a list of three numbers from 0 to 1, separated by forward slashes. The numbers represent the amount of red, green and blue in the resulting colour, e.g. 0.5/1/0.8.

Colour Max Level

Relevant only when **Mode** is set to **Preview**. The colours assigned to the data intervals are interpolated from **Colour Min Level** to **Colour Max Level**. Either choose a pre-set colour, or choose **Custom Rgb** and set **Colour Max Rgb**.

Colour Max Rgb

Relevant only when **Mode** is set to **Preview** and **Colour Max Level** is set to **Custom Rgb**. The colour is specified as a list of three numbers from 0 to 1, separated by forward slashes. The numbers represent the amount of red, green and blue in the resulting colour, e.g. 0.5/1/0.8.

Marker

Relevant only when **Mode** is set to **Preview**. Specifies which marker symbol is used to plot the data points. See the table of possible symbols in the MAGICS Users Guide, ChapterXII - Symbol Plotting - page 47.

Min Level Marker Size

Relevant only when **Mode** is set to **Preview**. Specifies the minimum size of the markers in cm. The markers used in the different intervals will be scaled between **Min Level Marker Size** and **Max Level Marker Size**.

Max Level Marker Size

Relevant only when **Mode** is set to **Preview**. Specifies the maximum size of the markers in cm. The markers used in the different intervals will be scaled between **Min Level Marker Size** and **Max Level Marker Size**.

Actions on the GeoTools icon

The actions available for the GeoTools icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate*, *Delete* and *Output*. Those requiring some information are :

Execute - this action, depending on the setting of **Mode**, plots a preview of the input data to the screen, plots a histogram of the data to the screen, or creates a new Symbol Plotting icon.

Visualise - performs the same actions as *Execute*.

Examine - in **Preview** mode, this action allows you to examine the contents of the data. It brings the data into an ASCII text editor for examination.

Save - in **Preview** mode, this action allows you to save the geopoints data as an ASCII file.

STATION



This icon allows you to specify a geographical location which may or may not coincide with a meteorological station. It is also equipped with a search facility applicable to the ECMWF's station database.

If the location is a meteorological station, you can use the **Stations** tool to search the database of stations. This tool is accessible from the **Tools** menu of any desktop. The search can be carried out by Identifier, WMO block, Position or Area. Note that the search results are presented in the form of Station icons ready to use or store in a folder, thus avoiding the use of the Station icon editor window.

For details of how to use the **Stations** search tool please refer to "Stations" in "The Metview Tools" on page I- 133.

The macro language equivalent is `stations()`

The Station Editor

Search Stations Database

Selects the type of station location to be entered. **Yes - Wmo Station** allows you to choose a station from the WMO station database. **No - Location** allows you to choose instead an arbitrary location. Note that in previous versions of Metview this field was labelled 'Station Type'.

Search Key

Specifies which parameter should be used in the search of the stations database. Only available for **Search Stations Database** set to **Yes - Wmo Station**. Options are **Name**, **Ident**, **WMO Block**, **Position** and **Area**. Each option when selected activates the parameter of the same name that follows in the editor.

Name

Specifies the location name. If **Search Stations Database** is set to **No - Location** you are dealing with an arbitrary location and you may enter any name of your choice. If **Search Stations Database** is set to **Yes - WMO Station** this should be the name of the required station. You need the exact name of the station so you should be careful with the spelling which is in accordance to WMO. This tries to match the original language spelling, hence an english speaking user should not look for Copenhagen or Oporto, but rather Kobenhavn and Porto.

If you are in doubt about the spelling or know only part of the name you can enter a guess string (e.g. "kobe" or "port") and hit return. Then click the station assist button to the left of the text field - you will see a list of stations whose name starts with the guess string. Click the one you want and its name will replace the guess string in the input field.

You can use the station assist button directly - it launches the Stations database search tool. From the stations returned by your search, click the one you want and its name will appear in the input field. For details on the search tool see "Stations" in "The Metview Tools" on page I- 133.

Position

Specifies the exact geographical coordinates of the location of interest. Enter latitude and longitude separated by a "/". To choose a location interactively use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42). If **Search Stations Database** is set to **No - Location** you specify the coordinates of an arbitrary location. If **Search Stations Database** is set to **Yes -WMO Station** you specify the location of a meteorological station.

It may happen that you do not know the exact coordinates of the station. In this case, you can specify a geographical tolerance in the following parameter, **Threshold**. If you need to use the Stations database search tool, run it from the main User Interface or from one of the other parameter's station assist button (which you must select in **Search Key**).

Height

Specifies the height of the station. If **Search Stations Database** is set to **No - Location** then you can specify the height here; otherwise the station's height is retrieved automatically from the database.

Threshold

Specifies a geographical tolerance in degrees for the coordinates of a meteorological station specified in Position. This is only available for **Search Stations Database** set to **Yes - WMO Station** and **Search Key** set to **Position**

Ident

Specifies the station's WMO numerical identifier which is a 5 digit number. If you are in doubt about the identifier, use the assist button. This is a station assist button and it launches the Stations database search tool. To operate the search tool see "Stations" in "The Metview Tools" on page I-133. Only available if **Search Stations Database** is set to **Yes - WMO Station** and **Search Key** to **Ident**

WMO Block

Specifies the station's WMO block which is a 2 digit number corresponding to a geographical area. If you are in doubt about the WMO block, use the assist button. This is a station assist button and it launches the Stations database search tool. To operate the search tool see "Stations" in "The Metview Tools" on page I- 133. Only available if **Search Stations Database** is set to **Yes - WMO Station** and **Search Key** to **Wmo Block**

Area

Specifies the coordinates of an area where meteorological stations of interest are located. Enter coordinates (lat/long) of an area separated by a "/" (top left lat and long, bottom right lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42). Only available if **Search Stations Database** is set to **Yes - WMO Station** and **Search Key** to **Area**. If you need to use the Stations database search tool, run it from the main User Interface or from one of the other parameter's station assist button (which you must select in **Search Key**)

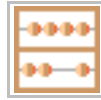
Actions on the Station Icon

The actions available for the Station icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate*, *Delete* and *Output*

Execute - this action retrieves the station coordinates from the Station Database. The station location is left in a cache, so that subsequent visualisations happen much faster.

Visualise - this action visualises the station location, using the default display window. The station location is left in a cache, so that subsequent visualisations happen much faster. For details on visualisation see "Visualisation - an Overview" on page I- 62.

SIMPLE FORMULA



This icon allows you to specify formulas of one of the following types :

- operations between one GRIB/geopoints icon and a scalar
- operations between two GRIB/geopoints icons
- functions of one or two GRIB/geopoints icons or of a GRIB/geopoints icon and a scalar value

For more complex operations please use a related icon named Formula (see "Formula" on page III-75).

The result of the operation returned by Simple Formula can be either :

- GRIB data - depending on the option specified in the icon menu, the resulting GRIB data can be visualised, dropped, examined or saved as a GRIB file icon on your desktop
- Geopoints data - depending on the option specified in the icon menu, the resulting geopoints data can be visualised, dropped or saved as a geopoints file icon on your desktop

There is no macro language equivalent. Specify the icon contents as macro functions and operations. If you are unsure about the exact translation, drop the prepared Simple Formula icon inside a Metview Macro editor window.

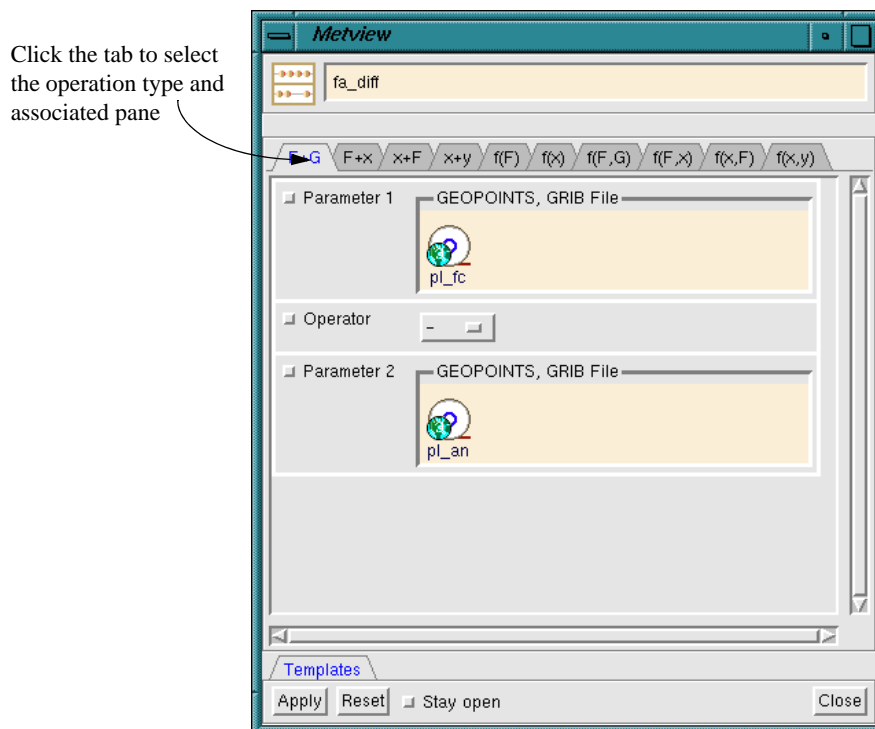


Figure III-3 A Simple Formula editor : it is divided in different input panes, one for each of the varieties of operation you can perform which you select by clicking its tab

The Simple Formula Editor

Edit/create a Simple Formula icon to obtain the editor window pictured in Figure III-3. The Simple Formula editor belongs to a type of editor known as *family editor* which allows related sets of input specifications to be grouped in the same icon editor, each in its own tabbed pane. The available working input panes are :

- F+G - operations between two GRIB/geopoints icons
- F+x - operations between GRIB/geopoints icon and scalar
- x+F - operations between scalar and GRIB/geopoints icon
- f (F) - functions of GRIB/geopoints icon
- f (F , G) - functions of two GRIB/geopoints icons
- f (F , x) - functions of GRIB/geopoints icon and scalar
- f (x , F) - functions of scalar and GRIB/geopoints icon

To operate with this icon first you need to select the broad type of operation you want to carry out (operation between two GRIB/geopoints icons, between GRIB/geopoints icon and scalar, etc.). You do this by clicking the corresponding tab. Each input pane contains a different set of parameters, but with common names and playing similar roles

Parameter

Specifies a single input element to a formula. It uses either a text field (to accept scalars) or an icon field (to accept a GRIB/geopoints icon) depending on the formula required

Parameter 1

Specifies the first input element of a formula. It uses either a text field (to accept scalars) or an icon field (to accept a GRIB/geopoints icon) depending on the formula required

Parameter 2

Specifies the second input element of a formula. It uses either a text field (to accept scalars) or an icon field (to accept a GRIB/geopoints icon) depending on the formula required

Operator

Specifies the operator to be used in the formula. Options are :

+	Addition	-	Subtraction
*	Multiplication	/	Division
^	Power		
>	Larger Than	<	Smaller Than
>=	Larger or Equal	<=	Smaller or Equal
=	Equal	<>	Not Equal
and	Conjunction	or	Disjunction

remember that the logical operators (last 4 lines) return data which is either 1 (result is true) or 0 (result is false)

Function

Specifies the function to be applied to the input defined in **Parameter** or in **Parameter 1** and **Parameter 2**. The available options depend on which of the panes you are in :

In the **f(F)** input pane the available options are :

- - - negative of the input
- **not** - negation of input : returns 0 where input values are non-zero, 1 otherwise.
- **sgn** - sign of the input : -1 where input is negative, 1 where positive, 0 where null.
- **int** - integer part of the input.
- **abs** - absolute part of the input.
- **sqrt** - square root of the input.
- **log** - natural logarithm of the input.
- **log10** - logarithm base 10 of the input.
- **exp** - exponential of the input.
- **sin** - sine of input (must be in radians).
- **cos** - cosine of input (must be in radians).
- **tan** - tangent of input (must be in radians).
- **asin** - arc sine of input (must be in radians).
- **acos** - arc cosine of input (must be in radians).
- **atan** - arc tangent of input (must be in radians).
- **coslat** - cosine of the latitude of the input grid points
- **sinlat** - sine of the latitude of the input grid points
- **count** - the number of fields within the input.
- **mean** - mean field of the input. Apply to GRIB input only.
- **rms** - root mean square field of the input. Apply to GRIB input only.
- **stdev** - standard deviation field of the input. Apply to GRIB input only.
- **sum** - sum field of the input
- **var** - variance field of the input. Apply to GRIB input only.

Note that some options apply only to GRIB icon input.

In the **f(F,G)** input pane the available options are :

- **max** - returns the maximum of the two input elements : a GRIB/geopoints of the maximum of the input GRIB/geopoints at each grid point or location. Output GRIB data contain the same number of fields as the maximum is taken on a field by field basis
- **min** - as above but returning the minimum
- **covar** - returns a GRIB field equal to the covariance of the input fieldsets. Apply only to GRIB input
- **div** - returns the integer result of division of **F** by **G**
- **merge** - returns a single GRIB fieldset composed of all the fields in the input GRIB fieldsets. Apply only to GRIB input
- **mod** - returns the result of **F** modulo **G**
- **interpolate** - returns a set of geopoints at the locations of the geopoints specified by **G**, with values calculated by interpolation at those points from the first field of fieldset **F**

In the $f(F, x)$ and $f(x, F)$ input panes the available options are :

- **max** - returns the maximum of the two input elements : a GRIB/geopoints of the maximum of the input GRIB/geopoints at each grid point or location and the specified location. Output GRIB data contain the same number of fields as the maximum is taken on a field by field basis
- **min** - as above but returning the minimum

Actions on the Simple Formula Icon

All actions from the icon menu are available for the Simple Formula icon. The ones requiring some explanation are :

Execute - this action reads/filters/retrieves the GRIB/geopoints data which is input to the specified formula and carries out the calculations. Output is kept in a cache on your workstation, but it is not automatically visualised. Apart from the changes of colour to the icon name, nothing else visible happens.

Visualise - this action carries out the visualisation of GRIB/geopoints data resulting from the specified formula, using the default display window. If the calculations have not been carried out before, the visualise action also performs them and stores the result in a cache. If the result of the formula is already in the cache, the visualisation happens much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62.

Save - this action allows you to save the data resulting from the specified formula as a GRIB or geopoints file. It works in the same way as for a MARS Retrieval icon : see "Saving MARS Retrieval Output (as Files)" on page III- 30.

Examine - this action allows you to examine the contents of the output file. Only works for GRIB output - see "Examining GRIB Data" on page III- 31 for details.

FORMULA



This icon allows you to specify any operation you wish involving any combination and number of input data icons, returning the result of the operation for visualising, saving or examining. Unlike its related icon Simple Formula this one is not limited in the operations it can perform

There is no macro language equivalent - you simply specify the same procedures in a macro program

The Formula Editor

The Formula icon editor is a plain text editor. The section "Working with Text Icon Editors" on page I- 51 provides full details on how to work with the Metview simple text editor, including how to use another text editor (vi, emacs, ...) in its place.

The desired operation is simply written in the text editor. The syntax for the operators is the same as that for the macro language but with the following restrictions :

- There is no assignment of the result to a variable, i.e. enter

```
z500 / 9.8
```

```
not
```

```
h = z500 / 9.8
```

- Macro-style comments (anything preceded by #) are not allowed
- The formula must be specified in a single line of code, but you can use a Formula icon as input to another Formula icon
- The input data (GRIB/geopoints/obs) must reside inside the Metview environment - all paths you specify are relative to the Metview root directory, e.g.

```
/mydata/T2 - 273.15
```

where the initial "/" stands for \$HOME/metview.

Provided these restrictions are obeyed, all macro functions and operations can be used in this icon.

Of course the variable names you use on the expression must correspond to existing icons on the Metview environment.

Actions on the Formula Icon

All actions from the icon menu are available for the Formula icon. The ones requiring some explanation are :

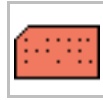
Execute - this action reads/filters/retrieves the GRIB/geopoints data which is input to the specified formula and carries out the calculations. Output is kept in a cache on your workstation, but it is not automatically visualised. Apart from the changes of colour to the icon name, nothing else visible happens.

Visualise - this action carries out the visualisation of GRIB/geopoints data resulting from the specified formula, using the default display window. If the calculations have not been carried out before, the visualise action also performs them and stores the result in a cache. If the result of the formula is already in the cache, the visualisation happens much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62.

Save - this action allows you to save the data resulting from the specified formula as a GRIB or geopoints file. It works in the same way as for a MARS Retrieval icon : see "Saving MARS Retrieval Output (as Files)" on page III- 30.

Examine - this action allows you to examine the contents of the output file. Only works for GRIB output - see "Examining GRIB Data" on page III- 31 for details.

MACRO



This icon is fully covered in Vol. II- Macro Language of this manual.

The Macro editor is essentially a simple text editor with extra functionality required for the writing, running and debugging of macro programs. For details on entering text into the macro editor, see "Working with Text Icon Editors" on page I- 51.

A related icon, Macro with Parameters which is used to provide a user interface to a macro program is found on page III-79.

MACRO PARAMETERS



This icon accepts a macro program as input and builds a user interface ready to accept user input. The user enters the required input in this interface, saves and on a given icon action, the embedded macro program is executed.

The input macro program must have the required code for the building of the user interface and parsing of the input

The Macro with Parameters Editor

The Macro Parameters editor is dynamic in that its layout and contents are determined by the user depending on what is specified in the macro program the user provides. The rationale is that the editor contents form a user interface accepting variable user input for execution by the macro provided. It is a circular process in that the Macro icon passed to the Macro Parameters specifies the user interface, the Macro Parameters builds it, accepts the input and passes it back to the Macro icon to be used in whatever calculations it specifies.

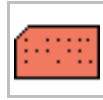
Full details of the process can be found in "Building a Macro User Interface" on page II- 105

Macro

Specifies the macro program which determines the layout and contents of the remainder of the editor.

This macro program must contain code to build the user interface and code to accept and parse the input (and obviously code to use it). On dropping a suitably coded macro program, a user interface is built. See also "Using Macro Parameters" on page II- 110 for information on macro coding of user interfaces.

NEDIT SYNTAX HIGHLIGHT



Note that this icon has now been replaced by a new version - see "NEDIT Syntax Highlight 2.0" on page III- 83.

This is a special macro that configures the text editor 'NEdit' for use as an alternative Metview macro editor with full syntax highlighting. See "Using alternative editors" on page I- 52 for details of alternative macro editors.

Note that if you cannot see this icon in the Macros drawer, then you must open the drawer, click-right and select 'Update this drawer...'

For more information on NEdit, see <http://www.nedit.org>. Note that the ECMWF is not responsible for information held on external sites.

NEDIT Syntax Highlight Details

When run, the macro will create a configuration file containing details of the Metview macro language, plus a set of default syntax highlighting styles. This file is then imported into NEdit whereupon the user is invited to save this new configuration via NEdit's **Preferences | Save Defaults** menu command. A backup of the original NEdit configuration file (`~/.nedit`) is made in `~/.nedit_metview_bak`.

A list of Metview macro's built-in functions is produced via the `dictionary()` macro command to be used by NEdit's syntax highlighting. This list, however, is not exhaustive, although it should become more complete in future versions of Metview. In the meantime, it is possible to add to this list once the macro has been run. This can be done by either manually editing the `.nedit` configuration file or by running NEdit, selecting **Preferences | Default Settings | Syntax Highlighting | Recognition Patterns...**, setting **Language Mode** to **Metview Macro** and adding the function names to the list specified by `BuiltinFunctions`, separating them with the vertical bar `'|'` character.

If the user's environment variable `'METVIEW_MACRO_EDITOR'` (or `'EDITOR'`) is not currently set to `'nedit'`, it must be in order to use NEdit as Metview's alternative text editor. See "Using alternative editors" on page I- 52 for details of using alternative macro editors.

NEdit will recognise a text file as being in Metview Macro format if it contains a comment character (`#`) within the first 200 characters and does not conform to any other file format's syntax. If NEdit fails to interpret a file as Metview Macro, set it manually using **Preferences | Language Mode**.

*Actions on the **NEDIT SYNTAX HIGHLIGHT** Icon*

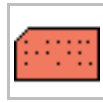
All actions from the icon menu are available for the NEDIT Syntax Highlight icon. The important ones are described below.

Execute - this action executes the macro and enables Metview macro syntax highlighting within NEdit.

Edit - this action, although not recommended, allows you to edit the macro, should you wish to make minor changes to its behaviour before executing it.

Output - this action allows you to view the output from the macro. If all went well on execution, the macro status should be green, and the output should contain no warnings or error messages.

NEDIT SYNTAX HIGHLIGHT 2.0



This is a special macro that configures the text editor ‘NEdit’ for use as an alternative Metview macro editor with full syntax highlighting and a number of useful features. See "Using alternative editors" on page I- 52 for details of alternative macro editors and "Using Customised NEdit as the Alternative Editor" on page I- 54 for more specific information on using NEdit as a macro editor.

Note that if you cannot see this icon in the Macros drawer, then you must open the drawer, click-right and select ‘Update this drawer...’

For more information on NEdit, see <http://www.nedit.org>. Note that the ECMWF is not responsible for information held on external sites.

NEDIT Syntax Highlight 2.0 Details

First you should ensure that NEdit will be used as your alternative macro editor:

```
echo $METVIEW_MACRO_EDITOR
```

You either want this environment variable to not be set to anything, or else you can set it manually to be `neditmv`; this is Metview’s default if it is not already set. The Metview startup script intercepts this value and converts it into a command-line string that will ensure that NEdit is started in Metview Macro language mode when launched from within the Metview environment. Note that you can supply a complete path to the editor’s executable if needed (for instance if you have a different version installed in a non-default directory). In this case, just make sure that you append ‘mv’ to the name of the file (which must be ‘nedit’). You may have to modify a shell startup script (eg `~/ .cshrc`) in order to make this change (if any is required) permanent. If a change to this environment variable is required, then you should restart Metview after making it.

Now *Execute* a copy of the ‘NEDIT Syntax Highlight 2.0’ macro that you have dragged to a Metview desktop. You will be presented with a dialogue box asking you which version of NEdit is installed on your system. If you do not know, type `nedit -version` on the command line to find out. Once you do know, you only need to indicate whether it is earlier than 5.4 or not. Version 5.4 has a feature called *calltips* which is not present in earlier versions; this affects the operation of the macro. The macro then modifies your NEdit configuration files - see below for more details if you are interested. Finally, you will be prompted from within a new NEdit window to save the new preferences; doing so will complete the initial phase of the installation.

You can now finish the setup procedure by following the instructions given here (for more detailed information, see "Updating NEdit’s Metview-specific files" on page I- 60). Start NEdit from within Metview - edit a macro and click on the pencil icon. From the **Macro** menu, run "**Macro | Refresh Lists, Calltips & Templates**" - the output will very probably list various macro files that were not documentable - that’s ok.

NEdit's configuration files

From version 5.4 onwards, NEdit uses an environment variable, `$NEDIT_HOME`, to point to a directory that contains the NEdit configuration files - these files are `nedit.rc` and `autoload.nm`. In previous versions of NEdit, the equivalents of these files were held in the user's root directory and were called `.nedit` and `.neditmacro` respectively. The setup macro icon takes these differences into account. Additionally, if NEdit version 5.4 is available on the system (ie if the user selects ">= 5.4" in the dialogue box) but the user has not yet defined `$NEDIT_HOME`, a new default NEdit home directory, `~/.nedit_home`, is created and the user's settings copied there. When Metview is next started, if `$NEDIT_HOME` is not defined, it checks for the existence of this directory and temporarily sets `$NEDIT_HOME` to point to it if it exists. The reason for duplicating NEdit's settings like this is that it is possible that more than one version of NEdit is available to the user (for instance if multiple operating systems are used, each with a different version of NEdit installed); this system allows all versions of NEdit to continue to work. If the user already has `$NEDIT_HOME` defined, then this procedure is not performed. If only an earlier version of NEdit is available, then only the 'old-style' configuration files are modified; `$NEDIT_HOME` is not used by earlier versions.

Changes to NEdit's configuration files

This section assumes the use of the 5.4 or later configuration files for illustrative purposes. A new NEdit *language mode* is created for Metview Macro, as well as a set of NEdit macros. After a backup has been made (`nedit_metview_bak`), the details are added to `nedit.rc`. The bulk of the macro code is written to a file `~/.metview_aux_files/nedit_aux_macros.nm` - this code is then only loaded into memory when needed, not every time that NEdit is started.

Additional customisations for NEdit

There are additional customisations that can be made in order to make NEdit a little easier to work with. Among these is the ability to set a fixed-width font for the calltips and function lists; this will create a better layout when using these features. To do this, you must edit your `~/.Xdefaults` file. Add the following lines to the end:

```
nedit*calltip.FontList: -adobe-courier-medium-r-normal--12-*-*-*-*-*
nedit*FontList: -adobe-courier-medium-r-normal--12-*-*-*-*-*
```

and type

```
xrdb .Xdefaults
```

Further modifications are possible - search the web or look on the NEdit home page.

Actions on the NEDIT SYNTAX HIGHLIGHT

2.0 Icon

All actions from the icon menu are available for the NEDIT Syntax Highlight 2.0 icon. The important ones are described below.

Execute - this action executes the macro and enables Metview macro syntax highlighting and other capabilities within NEdit.

Edit - this action, although not recommended, allows you to edit the macro, should you wish to make minor changes to its behaviour before executing it.

Output - this action allows you to view the output from the macro. If all went well on execution, the macro status should be green, and the output should contain no warnings or error messages.

AVERAGE DATA



This icon derives (and returns) a vertical average cross sectional data unit of upper air fields. For each upper air field, this average is taken along the North-South or East-West direction over a specified rectangular area and then interpolated horizontally along the direction perpendicular to the averaging direction with a spacing consistent with the resolution of the input GRIB data.

The average cross section data can be plotted or saved as a NetCDF data file (which gets a specific icon of course).

A related icon, Average View is used to provide average cross section plotting specifications. You can use it to visualise an average cross-section without explicitly deriving the average cross-section data - see "Average View" on page III- 139 and references therein.

The macro language equivalent is `xs_average()`

The Average Data Editor

Data

Specifies the data (GRIB icon) from which to derive the average profile. The input GRIB icon must specify a multi-level (pressure or model levels) upper air meteorological variable, in a latitude-longitude or Gaussian grid. If the input data is specified in model levels, you must include the parameter LNSP should you want the vertical axis of the plot in pressure levels rather than model levels when visualising the output. Note that the input fields should be on the same grid.

If more than one time and/or forecast step is contained in the GRIB icon, Average returns a set of average cross sections.

Area

Specifies the coordinates of the area over which the average profile is calculated. Enter coordinates (lat/long) of an area separated by a "/" (top left lat and long, bottom right lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42).

Direction

Specifies the direction along which the averaging of the variable is performed. Options are **North South** and **East West**. For **North South**, the averaging is weighted by $\cos(\text{latitude})$

Bottom Pressure

Specifies the lower limit of the cross section, in hPa

Top Pressure

Specifies the upper limit of the cross section, in hPa

Pressure Level Axis

Specifies the type of pressure axis - **Linear** or **Logarithmic**

Interpolate Values

Specifies whether to interpolate vertically the cross-section values into a regular set of interpolated vertical levels. Choosing not to interpolate returns an average cross section data matrix with values at the original levels.

Note - if you need to use the average cross section data in calculations further down the stream you should not interpolate.

Actions on the Average Data Icon

All actions from the icon menu are available for the Average Data icon. The ones requiring some explanation are :

Execute - this action reads/retrieves the input GRIB data and derives the average cross-section. Output is kept in a cache on your workstation, but it is not automatically visualised. Apart from the changes of colour to the icon name, nothing else visible happens.

Visualise - this action carries out the visualisation of the average cross-section data, using the default display window. If the data retrieval and calculations have not been carried out before, the visualise action also performs them and stores the result in a cache. If the average cross-section output is already in a cache, the visualisation happens much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62.

Save - this action allows you to save the data resulting from the specified formula as a NetCDF file. It works in the same way as for a MARS Retrieval icon (see "Saving MARS Retrieval Output (as Files)" on page III- 30) in that you get a save-output window where you can browse and specify a destination and name for the output data file. The output data file is in NetCDF format.

Examine - this action allows you to examine the contents of the generated NetCDF output. It launches a NetCDF data viewer with some capacity to investigate the resulting data.

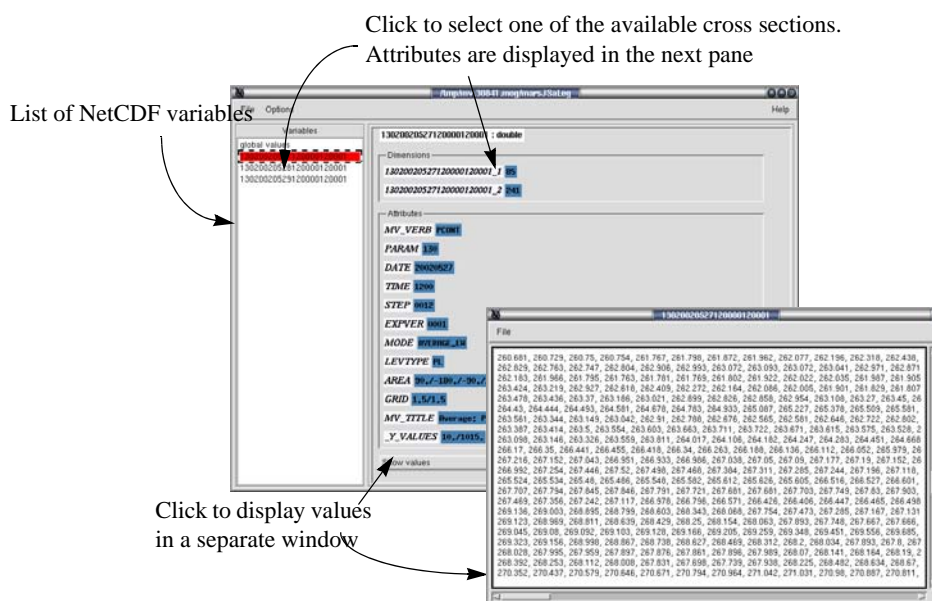


Figure III-4 The NetCDF viewer launched from an Examine action on an Average icon

CROSS-SECTION DATA



This icon derives (and returns) a vertical cross section data unit of upper air fields along a specified arbitrary transect line. For each upper air field, point values are interpolated along the transect line, with a spacing consistent with the resolution of the input GRIB data.

The cross section data can be plotted or saved as a NetCDF data file (which gets a specific icon of course).

A related icon, Cross-Section View is used to provide cross section plotting specifications. You can use it to visualise a cross-section without explicitly deriving the cross-section data - see "The View Concept in Visualisation" on page I- 65.

The macro language equivalent is `cross_sect ()`

The Cross-Section Data Editor

Data

Specifies the data (GRIB icon) from which to derive the cross-section profile. The input GRIB icon must specify a multi-level (pressure or model levels) upper air meteorological variable, in a latitude-longitude or Gaussian grid. If the input data is specified in model levels, you must include the parameter LNSP should you want the vertical axis of the plot in pressure levels rather than model levels when visualising the output.

If wind arrows are to be plotted, then the input data should include three-dimensional wind data, i.e. the u/v/w wind components should all be present.

If more than one time and/or forecast step is contained in the GRIB icon, Cross-Section returns a set of cross sections.

Line

Specifies the coordinates of a transect line along which the cross-section is calculated. Enter coordinates (lat/long) of a line separated by a "/" (easternmost lat and long, westernmost lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42).

Note that it is possible to define a line through either pole by describing the line's coordinates as follows. First, when specifying the latitudes of the two points, imagine that the latitude values go above 90 when you cross the North Pole and below -90 when you cross the South Pole (see Figure III-5 on page III-90). Next, if you wish a straight line, ensure that the two longitude values are the same as each other. An example demonstrates this. Say you wanted to defined a straight-line cross-section from 60S/25E to 60S/155W. This would be specified as -60/25/-120/25. The fact that one of the latitude values is below -90 indicates to Metview that a cross-section going through the South Pole is desired. Once this has been established, the fact that the two longitude values are identical tells Metview to use a straight line through the pole. If this is the intent, then only one unique longitude value is required, as the other one can be deduced. Giving Metview two different longitude values will cause a cross-section consisting of two curves to be produced.

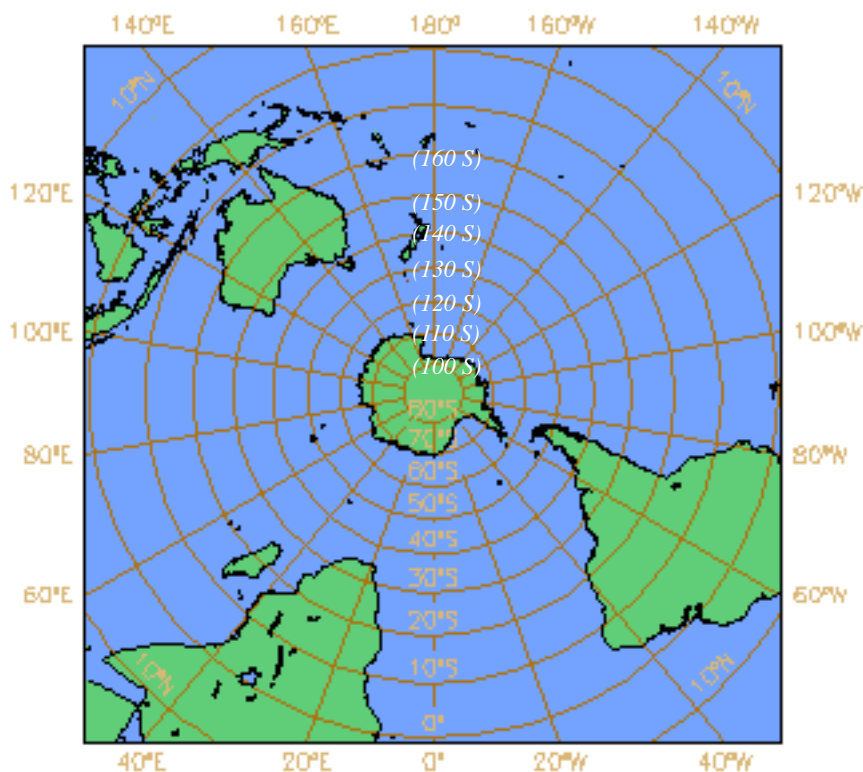


Figure III-5 Conceptual view of the South Pole when designing cross sections that go through the poles

Pressure Level Axis

Specifies the type of pressure axis - **Linear** or **Logarithmic**

Bottom Pressure

Specifies the lower limit of the cross section, in hPa

Top Pressure

Specifies the upper limit of the cross section, in hPa

Wind Parallel

Setting this option to **On** will produce a cross section plot of the projection of the wind onto the cross section plane (going from a 3D wind field to a 2D wind field projection). This is plotted using wind arrows. Valid values are **On/Off**.

Wind Perpendicular

Setting this option to **On** will produce a cross section plot showing the projection of the horizontal wind components onto the direction perpendicular to the cross section plane. The result is a one-dimensional quantity and is plotted with contour lines. Also produced is another cross section plot of the W component of the wind. Valid values are **On/Off**.

Wind Intensity

Setting this option to **On** will produce a cross section plot of the scalar wind intensity, plotted with contour lines. If one of the previous two parameters is **On**, then the intensity is that of the specified

projection. Also produced is another cross section plot of the W component of the wind. Valid values are **On/Off**.

Interpolate Values

Specifies whether to interpolate vertically the cross-section values into a regular set of interpolated vertical levels. Choosing not to interpolate returns a cross section data matrix with values at the original levels.

Note - if you need to use the average cross section data in calculations further down the stream you should not interpolate.

LnsP Param

Specifies the parameter number of the LnsP data, if you are using non-ECMWF data - ECMWF uses specific parameter numbers different from the WMO ones. Enter whichever parameter number is appropriate for your data.

U Wind Param

Specifies the parameter number of the U wind component data, if you are using non-ECMWF data - ECMWF uses specific parameter numbers different from the WMO ones. Enter whichever parameter number is appropriate for your data.

V Wind Param

Specifies the parameter number of the V wind component data, if you are using non-ECMWF data - ECMWF uses specific parameter numbers different from the WMO ones. Enter whichever parameter number is appropriate for your data.

W Wind Param

Specifies the parameter number of the W wind component data, if you are using non-ECMWF data - ECMWF uses specific parameter numbers different from the WMO ones. Enter whichever parameter number is appropriate for your data.

Actions on the Cross-Section Data Icon

All actions from the icon menu are available for the Cross-Section Data icon. The ones requiring some explanation are :

Execute - this action reads/retrieves the input GRIB data and derives the cross-section. Output is kept in a cache on your workstation, but it is not automatically visualised. Apart from the changes of colour to the icon name, nothing else visible happens.

Visualise - this action carries out the visualisation of the cross-section data, using the default display window. If the data retrieval and calculations have not been carried out before, the visualise action also performs them and stores the result in a cache. If the cross-section output is already in a cache, the visualisation happens much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62.

Save - this action allows you to save the resulting cross-section data as a NetCDF file. It works in the same way as for a MARS Retrieval icon (see "Saving MARS Retrieval Output (as Files)" on page III- 30) in that you get a save-output window where you can browse and specify a destination and name for the output data file. The output data file is in NetCDF format.

Examine - this action allows you to examine the contents of the generated NetCDF output. It launches a NetCDF data viewer which provides some capacity to investigate the resulting data - see "Actions on the Average Data Icon" on page III- 88 for an explaining figure

HOVMØLLER DATA



This icon derives (and returns) a Hovmøller diagram data unit along a specified arbitrary transect line or a rectangular area. This application can be used to display a two-dimensional graph with latitude or height as one axis and time as the other. If an input line is given, point values for each field are interpolated along the transect line, with a spacing consistent with the resolution of the input GRIB data. If a rectangular area is given, average values for each field are taken along the North-South or East-West direction.

The Hovmøller data can be plotted (using default plotting specifications) or saved as a NetCDF data file (which gets a specific icon of course).

The option 'expand Hovmøller' allows the production of Hovmøller diagrams to be computed incrementally. This could be useful, for example, if the input data is too large or the Hovmoeller diagram needs to be updated periodically (e.g. to produce a diagram operationally during a certain period of time).

A related icon, Hovmøller View is used to provide Hovmøller plotting specifications see "Hovmøller View" on page III- 145 and references therein.

The macro language equivalents are `hovmoeller_area()`, `hovmoeller_line()`, `hovmoeller_height()` and `hovmoeller_expand()` (depending on the Hovmøller type you specify).

The Hovmøller Data Editor

The Hovmøller Data editor belongs to a type of editor known as *family editor*, which allows related sets of input specifications to be grouped in the same icon editor, each in its own tabbed pane (like the Simple Formula icon - see Figure III-3 on page III-71). The available working input panes are:

- Line Hov - Provides input for Hovmøller diagrams derived from an input transect line.
- Area Hov - Provides input for Hovmøller diagrams derived from an input rectangular area.
- Height Hov - Provides input for Hovmøller diagrams derived from an input rectangular area and a set of levels.
- Expand Hov - Provides input for expanding Hovmøller diagrams that have been derived previously.

To operate with this icon first you need to select which output you want to produce by clicking the tab of the corresponding pane.

Line (Line Hov)

Specifies the coordinates of a transect line along which the Hovmøller diagram is calculated. Enter the coordinates (lat/long) of a line separated by a "/" (lat1/long1/lat2/long2); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42).

Area (Area Hov or Height Hov)

Specifies the coordinates of the area over which the Hovmøller diagram is calculated. Enter the coordinates (lat/long) of an area separated by a "/" (top left lat and long, bottom right lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42).

Data

Specifies the data (GRIB icon) from which to derive the Hovmøller diagram. The input GRIB icon must specify a time-series of a meteorological variable in a latitude-longitude or Gaussian grid. The assist button provides a default MARS Retrieval icon that you can modify to suit your requirements. See "MARS Retrieval" on page III- 13 for details of GRIB data specification.

If the selected option is **Area Hov** or **Line Hov** and if more than one parameter and/or level is contained in the GRIB icon, Hovmøller Data returns a set of Hovmøller diagrams. However, if the selected option is **Height Hov**, the GRIB icon should contain a set of levels for each parameter.

Average Direction (Area Hov)

Specifies the direction along which the averaging of the variable is performed. Options are **North South** and **East West**. For **North South**, the averaging is weighted by $\cos(\text{latitude})$.

Time Axis Direction

Specifies the direction along which the time-series is presented in the diagram. Options are **Forward** and **Backward**. For **Forward**, the time-series starts from the earliest time, presented at the origin of the diagram. For **Backward**, the time-series starts from the latest time, presented at the origin of the diagram.

Height Axis Direction (Height Hov)

Specifies the direction along which the level-series is presented in the diagram. Options are **Forward** and **Backward**. For **Forward**, the level-series starts from the smallest numerical value, presented at the origin of the diagram. For **Backward**, the level-series starts from the biggest numerical value, presented at the origin of the diagram.

Swap Axes

By default, the definition of the **Axis Orientation** value (**Vertical** or **Horizontal**) of parameters **Time Axis**, **Geo Axis** and **Geo2 Axis** follows pre-defined rules - see specific description in each of these three parameters. However, if **Swap Axes** is set to **Yes** then the **Axis Orientation** value for each of these parameters will be the opposite of that determined by the pre-defined rules.

Time Axis

Specifies the characteristics of the time-series axis to be used in the plotting of the data. The icon field accepts only icons of type Axis - see "Axis" on page III- 225. The following Axis parameters are defined by the application; therefore, users cannot change their values directly: **Axis Type** (set to **Date**), **Axis Date Min Value** and **Axis Date Max Value** (set according to the input fieldset), and **Axis Orientation**.

The **Axis Orientation** parameter is defined according to the following rules:

- a) if the selected input pane is **Line Hov**, the value is set to **Horizontal**;
- b) if the selected input pane is **Area Hov** and the **Average Direction** is **East West**, the value is set to **Horizontal**
- c) if the selected input pane is **Area Hov** and the **Average Direction** is **North South**, the value is set to **Vertical**.
- d) if the selected input pane is **Height Hov**, the value is set to **Horizontal**.

However, it is possible to change the **Axis Orientation** value from its calculated default by setting parameter **Swap Axes** to **Yes**; this option is not available for **Height Hov**.

Geo Axis

Specifies the characteristics of the geographical axis to be used in the plotting of the data. The icon field accepts only icons of type Axis - see "Axis" on page III- 225. The following Axis parameters are defined by the application; therefore, users cannot change their values directly: **Axis Min Value** and **Axis Max Value** (set according to the values given by parameters **Line** or **Area**), **Axis Tick Label Type** and **Axis Orientation**. The **Axis Tick Label Type** parameter is defined according to the following rules:

- a) if the selected input pane is **Line Hov**, the value is set to **Latitude**
- b) if the selected input pane is **Area Hov** and the **Average Direction** is **East West**, the value is set to **Latitude**
- c) if the selected input pane is **Area Hov** and the **Average Direction** is **North South**, the value is set to **Longitude**.

Moreover, the **Axis Orientation** parameter is defined according to the following rules:

- a) if the selected input pane is **Line Hov**, the value is set to **Vertical**
- b) if the selected input pane is **Area Hov** and the **Average Direction** is **East West**, the value is set to **Vertical**
- c) if the selected input pane is **Area Hov** and the **Average Direction** is **North South**, the value is set to **Horizontal**.

However, it is possible to change the **Axis Orientation** value from its calculated default by setting parameter **Swap Axes** to **Yes**.

Geo2 Axis (Line Hov)

Specifies the characteristics of the geographical axis to be used in the plotting of the data. The icon field accepts only icons of type Axis - see "Axis" on page III- 225. The following Axis parameters are defined by the application; therefore, users cannot change their values directly: **Axis Min Value** and **Axis Max Value** (set as the minimum and maximum longitude values given in the input parameter **Line**), **Axis Tick Label Type** (set as **Longitude**) and **Axis Orientation**.

The **Axis Orientation** parameter is set to **Vertical** unless **Swap Axes** is set to **Yes**, in which case the value will be set to **Horizontal**.

Height Axis (Height Hov)

Specifies the characteristics of the level-series axis to be used in the plotting of the data. The icon field accepts only icons of type Axis - see "Axis" on page III- 225. The following Axis parameters are defined by the application; therefore, users cannot change their values directly: **Axis Min Value** and **Axis Max Value** (set according to the input fieldset), **Axis Tick Label Type** and **Axis Orientation**.

Resolution

Used to interpolate the data onto a regular grid, and applies to both the horizontal and vertical axes where appropriate. This parameter is essential for creating a Hovmøller diagram from satellite data.

Netcdf Data (Expand Hov)

Specifies the data (Netcdf icon) from which to expand the Hovmoeller diagram.

Netcdf Path (Expand Hov)

Specifies the Netcdf file path and name. Type the full file path and name. Alternatively, parameter **Netcdf Data** can be used.

Data Path (Expand Hov)

Specifies the Grib file path and name. Type the full file path and name. Alternatively, parameter **Data** can be used.

Actions on the Hovmøller Data Icon

All actions from the icon menu are available for the Hovmøller Data icon. The ones requiring some explanation are :

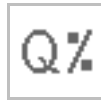
Execute - this action reads/retrieves the input GRIB data and derives the Hovmøller Diagram. Output is kept in a cache on your workstation, but it is not automatically visualised. Apart from the changes of colour to the icon name, nothing else visible happens.

Visualise - this action carries out the visualisation of the Hovmøller data, using the default display window. If the data retrieval and calculations have not been carried out before, the visualise action also performs them and stores the result in a cache. If the Hovmøller data output is already in a cache, the visualisation happens much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62.

Save - this action allows you to save the resulting Hovmøller data as a NetCDF file. It works in the same way as for a MARS Retrieval icon (see "Saving MARS Retrieval Output (as Files)" on page III- 30) in that you get a save-output window where you can browse and specify a destination and name for the output data file. The output data file is in NetCDF format.

Examine - this action allows you to examine the contents of the generated NetCDF output. It launches a NetCDF data viewer with some capacity to investigate the resulting data - see "Actions on the Average Data Icon" on page III- 88 for an explaining figure.

PERCENTILE



This icon allows you to compute a set of percentiles of a given input fieldset. A percentile is a number such that the given percentage of a distribution is equal to or below it. For instance, the median is the 50th percentile - 50% of the values are equal to or below it.

The generated fields can simply be visualised or saved as GRIB files.

The macro language equivalent is `percentile()`.

The Percentile Data Editor

Data

Specifies the data required for the application. This can be any icon returning GRIB data (e.g. MARS Retrieval, GRIB Filter, Formula, Simple Formula). The icon field assist button provides a tailor made MARS request in case you need some guidance in the data specification.

Source

Specifies the GRIB file path and name. Type the full file path and name. Alternatively, parameter **Data** can be used.

Percentiles

Specifies a list of values from 0 to 100 separated by a forward slash "/". The total number of values should be equal or smaller than the number of input fields, as each output field (which corresponds to each percentile value) will use the same input field structure. This will save computer resources and optimize the processing time.

Interpolation

Specifies the interpolation method used to compute the percentiles: **Nearest Neighbour** or **Linear**. Given a list of numbers V , the algorithm used to compute a percentile is the following.

1 - Compute the rank (R) of a P th percentile. This is done using the following formula:

$$R = P/100 \times (N + 1)$$

where P is the desired percentile and N is the number of input fields. If R were an integer, the P th percentile would be the number with rank R . When R is not an integer, the P th percentile is computed by interpolation as follows.

2 - If the interpolation method is **Nearest Neighbour**, the following equation is used:

$$P_{th} = V[\text{int}(R + 0.5)]$$

3 - If the interpolation method is **Linear**, the following steps are used:

3.1. Define IR as the integer portion of R .

3.2. Define FR as the fractional portion of R .

3.3. Find the scores with Rank IR and with Rank IR + 1, e.g. V[IR] and v[IR+1].

3.4. Interpolate by multiplying the difference between the scores by FR and add the result to the lower score, e.g.

$$P_{th} = FR * (v[IR+1] - V[IR]) + V[IR]$$

Actions on the PercentileIcon

The actions available for the Percentile icon are *Execute*, *Visualise*, *Edit*, *Examine*, *Save*, *Duplicate*, and *Delete*. The ones requiring some explanation are :

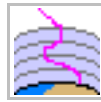
Execute - this action retrieves the input data and derives the specified output. The data is put in a cache, so that subsequent actions happen much faster.

Visualise - this action carries out the visualisation of the derived field, using the default display window. If not visualised / executed before it also carries out the retrieval of the input data and derivation of the output. The data is left in a cache, so that subsequent actions happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

Save - this action allows you to save the derived field as a GRIB file. It works in the same way as for a MARS Retrieval icon : see "Saving MARS Retrieval Output (as Files)" on page III- 30.

Examine - this action allows you to examine the contents of the derived field in the same way as you would a GRIB icon : see "Examining GRIB Data" on page III- 31.

VERTICAL PROFILE DATA



This icon derives (and returns) a vertical profile data unit of upper air fields for a particular point location (or small area). For each upper air field, values are interpolated at the point location (or integrated over the small area).

The vertical profile data can be plotted or saved as a NetCDF data file (which gets a specific icon of course).

A related icon, Vertical Profile View is used to provide vertical profile plotting specifications. You can use it to visualise a vertical profile without explicitly deriving the vertical profile data - see "The View Concept in Visualisation" on page I- 65.

The macro language equivalent is `vert_prof()`.

The Vertical Profile Data Editor

Data

Specifies the data (GRIB icon) from which to derive the vertical profile data. The input GRIB icon must specify a multi-level (pressure or model levels) upper air meteorological variable, in a latitude-longitude or Gaussian grid. If the input data is specified in model levels, you must include the parameter LNSP should you want the vertical axis of the plot in pressure levels rather than model levels when visualising the output.

If more than one time and/or forecast step is contained in the GRIB icon, Vertical Profile returns a set of vertical profiles.

Input Mode

Specifies whether to derive the vertical profile for a **Point** or an **Area**. **Nearest Gridpoint** will take the nearest gridpoint to the point specified. **Area 2** is an enhanced version of **Area**. It is more accurate and will work with a greater variety of projections. It should be used in preference to **Area**, which is included simply for backwards compatibility.

Point

Specifies the coordinates of the point for which the vertical profile is calculated. Enter coordinates (lat/long) of a point separated by a "/" ; alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42

Area

Specifies the coordinates of the area over which the averages composing the vertical profile are calculated. Enter coordinates (lat/long) of an area separated by a "/" (top left lat and long, bottom right lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42

Bottom Pressure

Specifies the lower limit of the vertical profile, in hPa

Top Pressure

Specifies the upper limit of the vertical profile, in hPa

Pressure Level Axis

Specifies the type of pressure axis - **Linear** or **Logarithmic**

Actions on the Vertical Profile Data Icon

All actions from the icon menu are available for the Vertical Profile Data icon. The ones requiring some explanation are :

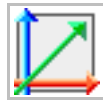
Execute - this action reads/retrieves the input GRIB data and derives the vertical profile. Output is kept in a cache on your workstation, but it is not automatically visualised. Apart from the changes of colour to the icon name, nothing else visible happens.

Visualise - this action carries out the visualisation of the vertical profile data, using the default display window. If the data retrieval and calculations have not been carried out before, the visualise action also performs them and stores the result in a cache. If the vertical profile output is already in a cache, the visualisation happens much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62.

Save - this action allows you to save the resulting vertical profile data as a NetCDF file. It works in the same way as for a MARS Retrieval icon (see "Saving MARS Retrieval Output (as Files)" on page III- 30) in that you get a save-output window where you can browse and specify a destination and name for the output data file. The output data file is in NetCDF format.

Examine - this action allows you to examine the contents of the generated NetCDF output. It launches a NetCDF data viewer which provides some capacity to investigate the resulting data - see "Actions on the Average Data Icon" on page III- 88 for an explaining figure.

VECTORS



This icon returns a vector from user supplied data of vector X and Y components (cartesian coordinates) or intensity/direction (polar coordinates) with the option of colouring the vectorial representation (arrows) according to the magnitude of a user supplied scalar quantity. The classic example is to plot a wind field from (U,V) components coloured according to Temperature.

The macro language equivalents are `vector_field()`, `polar_field()`, `colour_vector_field()` and `colour_polar_field()` - depending on what you specify.

The Vectors Editor

The Vectors editor belongs to a type of editor known as *family editor* which allows related sets of input specifications to be grouped in the same icon editor, each in its own tabbed pane (like the Simple Formula icon - see Figure III-3 on page III-71). The available working input panes are :

Vector Field - To derive a vector field from two arbitrary scalar fields which act as U and V components

Polar Field - To derive a vector field from two arbitrary scalar intensity (ρ) and direction (θ) fields

Colour Vector Field - To derive a vector field from two scalar u and v component fields coloured according to the value of a third scalar field

Colour Polar Field - To derive a vector field from two arbitrary scalar intensity and direction fields coloured according to the value of a third scalar field

To operate with this icon first you need to select which type of vector field you want to create by clicking the tab of the corresponding pane. Each input pane contains a different set of parameters, but with common names and playing similar roles.

U Component

Specifies the field to be used as the vector field U component. The parameter accepts any GRIB icon as input

V Component

Specifies the field to be used as the vector field V component. The parameter accepts any GRIB icon as input

Intensity

Specifies the field to be used as the vector field intensity component. The parameter accepts any GRIB icon as input

Direction

Specifies the field to be used as the vector field direction component. The parameter accepts any GRIB icon as input

Intervals

Specify a list of numbers separated by a forward slash "/". The numbers must be consistent with the field chosen as the colouring field - e.g. if the colouring field is temperature, the numbers must be Kelvin temperatures. If you do not specify this list, Metview assigns the chosen colours equally across the range of values of the colouring field.

The number of intervals defined by the numerical scale should be equal or smaller than the number of colours in the colour list, as each interval will correspond to a colour.

Colour List

Specifies a list of colours to be applied in the colouring of the vector field. Specify as many (or more) colours as the number of numerical intervals you have defined in **Intervals**. Use the colour assist tool - see "Colour selection list" on page I- 40 for details.

Unit Velocity

Specifies the wind speed represented by a unit vector. The higher this value the shorter the wind arrows. If you set this to 0.0, MAGICS plots wind arrows by setting the unit vector to the maximum wind speed in the field.

Calm Below

Specifies wind speed below which wind is considered calm.

Colouring Field

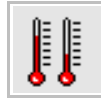
Specifies the field to be used as the colouring key. The parameter accepts any GRIB icon as input

Actions on the Vectors Icon

The actions available for the Vectors icon are *Visualise*, *Edit*, *Duplicate* and *Delete*.

Visualise - this action carries out the visualisation of the derived data, using the default plot window. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

RELATIVE HUMIDITY



This icon derives (and returns) the relative humidity from its input data. Inputs are temperature, specific humidity and LNSP (only required if the input data is specified in model levels). The generated fields can simply be visualised or saved as GRIB files.

Since Metview version 3.6-export and 3.6.1, the relative humidity is defined as in the IFS model output, i. e. with respect to water above 273.16 K, to ice below 250.16 K, and with respect to a mixture of both in between. See the IFS documentation at <http://www.ecmwf.int/research/ifsdocs/> for details.

The macro language equivalent is `relhum()`.

The Relative Humidity Editor

Data

Specifies the data required for the application. The icon field assist button provides a tailor made MARS request in case you need some guidance in the data specification.

Actions on the Relative Humidity Icon

The actions available for the Relative Humidity icon are *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate* and *Delete*.

Visualise - this action carries out the visualisation of the derived data, using the default plot window. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

POTENTIAL TEMPERATURE



This icon returns a GRIB field of one of the following :

- potential temperature
- equivalent potential temperature
- saturated equivalent potential temperature

Inputs are temperature, LNSP (only if the input data is specified in model levels) and humidity (only if calculating equivalent potential temperature). See Holton, 3rd Ed, Appendix D, eq. D-9, for details on the scientific background

The generated fields can be simply visualised or saved as GRIB files.

The macro language equivalents are `pott_m()`, `eqpott_m()` and `seqpott_m()`.

The Potential Temperature Editor

The Potential Temperature editor belongs to a type of editor known as *family editor* which allows related sets of input specifications to be grouped in the same icon editor, each in its own tabbed pane (like the Simple Formula icon - see Figure III-3 on page III-71). The available working input panes are :

Potential Temperature - To derive a potential temperature field from temperature data (and LNSP if using model levels)

Equivalent Potential Temperature - To derive an equivalent potential temperature field from temperature and humidity data (and LNSP if using model levels)

Saturated Equivalent Potential Temperature - To derive a saturated equivalent potential temperature field from temperature data (and LNSP if using model levels)

For each of these panes, you can select one of two other panes :

Model Levels - Derives the output using model levels. Requires a LNSP field additional to the temperature (and humidity) input data.

Pressure Levels - Derives the output using pressure levels. Requires only the temperature (and humidity) input data.

To operate with this icon first you need to select which type of potential temperature field you want to create by clicking the tab of the corresponding pane. You also need to click the pane corresponding to the type of level (model or pressure) your input is in.

Each input pane contains a different set of parameters with the same names and identical roles :

LNSP

Specifies the LNSP field as a GRIB icon. Only available/required for the Model Levels sub-pane.

Temperature

Specifies the temperature field as a GRIB icon in either model or pressure levels.

Humidity

Specifies the specific humidity field as a GRIB icon in either model or pressure levels. Only available/required for the Equivalent Potential Temperature pane.

Actions on the Potential Temperature Icon

The actions available for the Potential Temperature Family icon are *Execute*, *Visualise*, *Edit*, *Examine*, *Save*, *Duplicate*, and *Delete*

Execute - this action retrieves the input data and derives the specified output. The data is put in a cache, so that subsequent actions happen much faster.

Visualise - this action visualises the derived field. If not visualised / executed before it also carries out the retrieval of the input data and derivation of the output. The data is left in a cache, so that subsequent actions happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual

Examine - this action allows you to examine the contents of the derived field in the same way as you would a GRIB icon : see "Examining GRIB Data" on page III- 31.

Save - this action allows you to save the derived field as a GRIB file. It works in the same way as for a MARS Retrieval icon : see "Saving MARS Retrieval Output (as Files)" on page III- 30

VELOCITYPOTENTIAL/STREAMFUNCTION



This icon returns a GRIB field of one of the following :

- velocity potential from input data of Divergence
- stream function from input data of Vorticity

The generated fields can be simply visualised or saved as GRIB files.

The macro language equivalents are `velpot()` and `streamfn()`.

The Velocity Potential / Stream Function Editor

The Velocity Potential/Stream Function editor belongs to a type of editor known as *family editor* which allows related sets of input specifications to be grouped in the same icon editor, each in its own tabbed pane (like the Simple Formula icon - see Figure III-3 on page III-71). The available working input panes are :

VelPot - To derive velocity potential fields from input of divergence

Streamfn - To derive streamfunction fields from input of vorticity

To operate with this icon first you need to select which output you want to produce by clicking the tab of the corresponding pane. In this icon the two input panes contain identical sets of input parameters :

Data

Specifies the data required for the application. The icon field assist button provides a tailor made MARS request in case you need some guidance in the data specification. The requested data is Divergence in the VelPot pane and Vorticity in the Streamfn pane. Both input data are specified in spherical harmonics

Truncation

Specifies the truncation to be applied to the spherical harmonics input data prior to conversion to lat/long (see "Resol" on page III- 25).

Smoothing

Specifies whether to apply spatial smoothing to the spherical harmonics prior to transformation to grid points. This operation is performed after the truncation specified in **Truncation**. The smoothing filter is of the form :

$$sn = \exp(-(n*(n+1)/fltc*(fltc+1))*mfltextp)$$

this is roughly equivalent to a $\nabla^2 \times mfltextp$ operator in grid point space.

Fltc

Specifies the value of the parameter `fltc` to be used in the smoothing filter. Only available if **Smoothing** set to **Yes**. The default value is 19.4

Mfltxp

Specifies the value of the parameter `mfltxp` to be used in the smoothing filter. Only available if **Smoothing** set to **Yes**. The default value is 2, roughly equivalent to a ∇^4 operator in grid point space.

Actions on Velocity Potential/Stream Function Icon

The actions available for the Velocity Potential/Streamfunction icon are *Execute*, *Visualise*, *Edit*, *Examine*, *Save*, *Duplicate* and *Delete*

Execute - this action retrieves the input data and derives the specified output. The data is put in a cache, so that subsequent actions happen much faster.

Visualise - this action visualises the derived field. If not visualised / executed before it also carries out the retrieval of the input data and derivation of the output. The data is left in a cache, so that subsequent actions happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual

Examine - this action allows you to examine the contents of the derived field in the same way as you would a GRIB icon : see "Examining GRIB Data" on page III- 31.

Save - this action allows you to save the derived field as a GRIB file. It works in the same way as a MARS Retrieval icon : see "Saving MARS Retrieval Output (as Files)" on page III- 30.

ROTATIONALWIND/DIVERGENTWIND



This icon returns a GRIB field of one of the following :

- divergent wind vectors from input data of Divergence
- rotational wind vectors from input data of Vorticity

The generated fields can be simply visualised or saved as GRIB files.

The macro language equivalents are `divrot()` and `divwind()`.

The Rotational Wind / Divergent Wind Editor

The RotationalWind/DivergentWind editor belongs to a type of editor known as *family editor* which allows related sets of input specifications to be grouped in the same icon editor, each in its own tabbed pane (like the Simple Formula icon - see Figure III-3 on page III-71). The available working input panes are :

RotWind - To derive rotational wind vector fields from input of vorticity

DivWind - To derive divergent wind vector fields from input of divergence

To operate with this icon first you need to select which output you want to produce by clicking the tab of the corresponding pane. In this icon the two input panes contain identical sets of input parameters :

Data

Specifies the data required for the application. The icon field assist button provides a tailor made MARS request in case you need some guidance in the data specification. The requested data is Divergence in the Divwind pane and Vorticity in the Rotwind pane. Both input data are specified in spherical harmonics

Truncation

Specifies the truncation to be applied to the spherical harmonics input data prior to conversion to lat/long (see "Resol" on page III- 25).

Smoothing

Specifies whether to apply spatial smoothing to the spherical harmonics prior to transformation to grid points. This operation is performed after the truncation specified in **Truncation**. The smoothing filter is of the form :

$$sn = \exp \left(-(n*(n+1)/fltc*(fltc+1))^{**mfltexp} \right)$$

this is roughly equivalent to a $\nabla^2 \times mfltexp$ operator in grid point space

Fltc

Specifies the value of the parameter `fltc` to be used in the smoothing filter. Only available if **Smoothing** set to **Yes**. The default value is 19.4

Mfltexp

Specifies the value of the parameter `mfltexp` to be used in the smoothing filter. Only available if **Smoothing** set to **Yes**. The default value is 2, roughly equivalent to a ∇^4 operator in grid point space

Actions on RotationalWind/DivergentWind Icon

The actions available for the Rotational Wind / Divergent Wind icon are *Execute*, *Visualise*, *Edit*, *Examine*, *Save*, *Duplicate* and *Delete*

Execute - this action retrieves the input data and derives the specified output. The data is put in a cache, so that subsequent actions happen much faster.

Visualise - this action visualises the derived field. If not visualised / executed before it also carries out the retrieval of the input data and derivation of the output. The data is left in a cache, so that subsequent actions happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

Examine - this action allows you to examine the contents of the derived field in the same way as you would a GRIB icon : see "Examining GRIB Data" on page III- 31.

Save - this action allows you to save the derived field as a GRIB file. It works in the same way as a MARS Retrieval icon : see "Saving MARS Retrieval Output (as Files)" on page III- 30.

CUSTOM METGRAM



This icon allows you to plot a time series of a given parameter for a specified location (arbitrary or meteorological station).

The macro language equivalent is `metgram()`.

Note that as of Metview 3.7.2, this icon has been replaced with Time Series (see "Time Series" on page III- 113).

The Metgram Editor

The input parameters for this icon are all other icons. These provide the elements required from the plot - a data icon, a station icon, and two icons for plotting characteristics specification.

Data

Specifies the data to be used in the plotting. This can be any icon returning GRIB or BUFR data (e.g. MARS Retrieval, GRIB Filter, Observation Filter, Formula, Simple Formula)

Station

Specifies the location at which the time series will be plotted. This can be a meteorological station part of the ECMWF data base or any arbitrary location of the user's choice. See "Station" on page III- 67

Vertical Axis

Specifies the characteristics of the vertical axis to be used in the plotting of the temporal profile. The icon field accepts only icons of icon Axis - see description in "Axis" on page III- 225

Attributes

Specifies the attributes of the graph (e.g. plotted line's type and format). The icon field accepts only icons of icon Graph - see description in "Graph" on page III- 221

Actions on the Metgram Icon

The actions available for the Metgram icon are *Visualise*, *Edit*, *Duplicate* and *Delete*

Visualise - this action carries out the visualisation of the plot resulting from the specified parameters, using the default curve display window. The data from which the values were derived is kept in a cache, so that subsequent visualisations happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

TIME SERIES



This icon allows you to plot a time series of a given parameter for a specified location (arbitrary or meteorological station) or averaged over an area. Once created, this icon should be visualised with a Curve View icon (see "Curve View" on page III- 147). If plotting more than two time series, it is a good idea to reduce the 'Subpage Y Length' parameter of the curve view in order to allow the legend to fit more easily into the plot.

The macro language equivalents are `fieldpoint_timeseries()`, `fieldmean_timeseries()` and `geopoint_timeseries()`.

The Time Series Editor

The Time Series editor belongs to a type of editor known as *family editor* which allows related sets of input specifications to be grouped in the same icon editor, each in its own tabbed pane (like the Simple Formula icon - see Figure III-3 on page III-71). The available working input panes are :

FieldPoint Timeseries- Derives time series from a point within fieldset data.

FieldMean Timeseries- Derives time series from an integrated area of fieldset data.

GeoPoint Timeseries - Derives time series from observation (geopoint) data.

To operate with this icon first you need to select which output you want to produce by clicking the tab of the corresponding pane. The parameter list below contains both panes' parameters with those unique to a given pane flagged.

The input parameters for this icon are all other icons. These provide the elements required from the plot - a data icon, a station icon, and an icon for plotting characteristics specification.

Data (FieldPoint Timeseries, FieldMean Timeseries)

Specifies the data to be used in the plotting. This can be any icon returning GRIB data (e.g. MARS Retrieval, GRIB Filter, Formula, Simple Formula).

Data (GeoPoint Timeseries)

Specifies the data to be used in the plotting. This can be any icon returning geopoints data (e.g. Observation Filter or a geopoints data icon).

Data (Bufr Timeseries)

Specifies the data to be used in the plotting. This can be any icon returning BUFR data (e.g. MARS Retrieval or Observation Filter).

Station (FieldPoint Timeseries)

Specifies the location at which the time series will be plotted. This can be a meteorological station part of the ECMWF data base or any arbitrary location of the user's choice. See "Station" on page III- 67. Multiple stations may be specified - however, each will be plotted with the same attributes.

Area (FieldMean Timeseries)

Specifies the area over which to integrate the data values.

Message (Bufr Timeseries)

Specifies the message number from which the data is to be used.

Name (Bufr Timeseries)

Specifies the name of the parameter to be plotted. By default, this is automatically retrieved from the BUFR message.

Scaling

Specifies the components of a linear scaling to be applied to the input data. The components are in the form of a list, comprising of the offset (applied first) followed by the multiplier.

Attributes

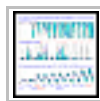
Specifies the attributes of the graph (e.g. plotted line's type and format). The icon field accepts only icons of icon Graph - see description in "Graph" on page III- 221

Actions on the Time Series Icon

The actions available for the Time Series icon are *Execute*, *Visualise*, *Edit*, *Duplicate* and *Delete*

Execute - this action prepares the data for plotting. In order to visualise the data, the icon must be dropped onto a Curve View window. The data from which the values were derived is kept in a cache, so that subsequent visualisations happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

METEOGRAM (ECMWF ONLY)



This icon allows you to plot a fixed set of meteorological parameters forecasts as a function of forecast time for a given location or set of locations (arbitrary or meteorological station). The following web page describes the meteorograms generated by this application: <http://www.ecmwf.int/contrib/pages/epsdoc.html>.

The macro language equivalent is `meteogram()`.

The Meteorogram Editor

Type

Specifies the type of meteorogram to be generated. The Epsgrams are derived from the Ensemble Prediction System, whereas the Metgrams are derived from the deterministic forecast model. The available types are **10 Days Epsgram**, **15 Days Epsgram**, **10 Days Metgram** and **10 Days Wave Epsgram**.

Station

Specifies the location at which to derive the meteorogram. Provide a Station icon with the location details (see "Station" on page III- 67).

Data Selection Type

Determines how the meteorogram data source is selected: **Latest** (default) will retrieve the latest meteorogram available; **Date** will allow the further selection of a specific date and time; **Local** allows the specification of a path to a local SPOT database (details of the database format are not provided here).

Date

Specifies the day on which the forecast is based (first day on the plot). The default value is -1 (yesterday), but you can use other formats, such as `YYMMDD` or `YYYY-MM-DD`. Only available if **Data Selection Type** is set to **Date**.

Forecast Run Time

Specify the forecast time (hours of the day) from either 00h or 12h. Only available if **Data Selection Type** is set to **Date**.

Experiment

Specifies the MARS experiment number from which the metgrams are to be plotted. You will only need to modify this parameter if you want to display data from a source other than the ECMWF model. Note that this parameter is a string, so for example '0001' is different from '1'.

Format

Specifies the output format (the default is PostScript). As this module uses Magics++, more formats are available than with other Metview modules.

Database

If not **latest**, then this parameter is taken to be the entire path to the database directory. Only available if **Data Selection Type** is set to **Local**.

Actions on the Meteogram Icon

The actions available for the Meteogram icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate* and *Delete*

Execute - this action generates a meteogram in a temporary directory.

Visualise - this action generates a meteogram in a temporary directory and visualises it with the preview program which has been set up for that particular output format.

Examine - this action generates a meteogram in a temporary directory and opens with the editor program which has been set up for that particular output format.

Save - this action generates a meteogram and asks where to save it to.

Macro Example of Meteogram Icon

The following example shows how to generate and save a meteogram from the Macro language:

```
london = stations (name : "London Weather Centre")

metgram_london = meteogram
(
  type          : "10_days_epsgram",
  format        : "pdf",
  station       : london
)

write ('metgram_london.pdf', metgram_london)
```

The valid values for `type` are derived from the possible values listed under "The Meteogram Editor" above, moved to lower case and with underscores replacing the spaces.

METGRAM MANAGER (DEPRECATED)



This icon is deprecated and will cease to function from 26th January 2010 with the increase in model resolution. Please use the Meteogram icon and macro function instead - see "Meteogram (ECMWF only)" on page III- 115.

TEPHIGRAM DATA



This icon prepares input to be supplied for plotting of tephigrams. You can prepare data for tephigrams based on observations or on model field data (two types depending on the required data). The plotting specification for tephigrams is provided by the Tephigram View icon - see "Tephigram View" on page III- 161.

The macro language equivalents are `pm_grib_tephi()`, `pm_bufrr_tephi()` and `pre1995gribtephi()` (depending on the tephigram type you specify).

The Tephigram Data Editor

The Tephigram Data editor belongs to a type of editor known as *family editor* which allows related sets of input specifications to be grouped in the same icon editor, each in its own tabbed pane (like the Simple Formula icon - see Figure III-3 on page III-71). The available working input panes are :

Pm Bufr Tephi - Provides input for tephigrams based on observation (BUFR) data.

Pm Grib Tephi - Provides input for tephigrams based on model field (GRIB) data post 4th April 1995

Pm Pre19950404Grib Tephi - Provides input for tephigrams based on model field (GRIB) data pre 4th April 1995

The two varieties of GRIB data tephigrams exist due to changes in the attributes of the input data - parameter Q changed to Gaussian Grid from Spherical Harmonics.

To operate with this icon first you need to select which output you want to produce by clicking the tab of the corresponding pane.

Data (Pm Bufr Tephi)

Specifies the data to be used in the plotting of a tephigram from observations data. You need upper air sounding data (VSNS - Vertical Sounding Non-Satellite in BUFR-speak) - PILOT observations (winds only) have not enough data for plotting a tephigram. The most straightforward action is to use the assist button which provides a default MARS Retrieval icon that you can modify to suit your requirements. See "MARS Retrieval" on page III- 13 for details of GRIB data specification

Data 1 / Data 2 (Pm Grib Tephi)

Specify the data to be used in the plotting of a tephigram from model field data. For GRIB tephigrams you need two MARS Retrievals, one for T (all model levels), plus LNRP (spherical harmonics) - specified in **Data 1** - another for Q (all model levels, in Gaussian Grid) - specified in **Data 2**. See "MARS Retrieval" on page III- 13 for details on GRIB data specification.

Olddata (Pm Pre19950404 Grib Tephi)

Specifies the data to be used in the plotting of a tephigram from model field data. Prior to 04/04/95, both T and Q were archived in spherical harmonics, hence you can specify the data with a single MARS icon. See "MARS Retrieval" on page III- 13 for details of GRIB data specification.

Station

Specifies the location for which the tephigram is to be plotted. In the Pm Bufr Teph sub-pane, your station must define a meteorological station that does full upper air soundings. For the other sub-panes, your station may represent any arbitrary point. See "Station" on page III- 67 for details of Station specification.

Line Colour

Specifies the colour of the Tephigram plot.

Line Thickness

Specifies the thickness of the lines, from 1 to 10.

Line Style

By default, the temperature line will be **SOLID** and the dewpoint line will be **DASH**. If this parameter is set, both lines will adopt the given style.

Flag Colour (Pm Bufr Teph)

Specifies the colour of the wind flag. Only available for the Pm Bufr Teph sub-pane.

Point (Pm Grib Teph/Pm Pre19950404 Grib Teph)

Specifies the way to calculate values at the point location for GRIB tephigrams. Options are:

- **Nearest** - using the data of the nearest grid point
- **Interpolate** - interpolate values from the surrounding grid points (default)
- **Area Average** - uses an area average, taking the area from the **Area** parameter

Area (Pm Grib Teph/Pm Pre19950404 Grib Teph)

Specifies the area over which to integrate when **Point** is set to **Area Average**.

Temperature Param (Pm Grib Teph/Pm Pre19950404 Grib Teph)

Specifies the parameter number of the temperature data, if you are using non-ECMWF data - ECMWF uses specific parameter numbers different from the WMO ones. Enter whichever parameter number is appropriate for your data.

Specific Humidity Param (Pm Grib Teph/Pm Pre19950404 Grib Teph)

Specifies the parameter number of the specific humidity, if you are using non-ECMWF data - ECMWF uses specific parameter numbers different from the WMO ones. Enter whichever parameter number is appropriate for your data.

U Wind Param (Pm Grib Teph/Pm Pre19950404 Grib Teph)

Specifies the parameter number of the U wind component data, if you are using non-ECMWF data - ECMWF uses specific parameter numbers different from the WMO ones. Enter whichever parameter number is appropriate for your data.

V Wind Param (Pm Grib Teph/Pm Pre19950404 Grib Teph)

Specifies the parameter number of the V wind component data, if you are using non-ECMWF data - ECMWF uses specific parameter numbers different from the WMO ones. Enter whichever parameter number is appropriate for your data.

LnsP Param (Pm Grib TephI/Pm Pre19950404 Grib TephI)

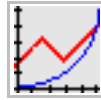
Specifies the parameter number of the LnsP data, if you are using non-ECMWF data - ECMWF uses specific parameter numbers different from the WMO ones. Enter whichever parameter number is appropriate for your data.

Actions on the Tephigram Data icon

The actions available for the Tephigram Data icon are *Edit*, *Duplicate* and *Delete*

This icon is only to be dropped inside a display window provided with a Tephigram view ("Tephigram View" on page III- 161) or used within macro.

CURVE



This icon plots user supplied data on a x-y axis in a variety of formats. It is available to provide some interactive equivalent but for serious usage you really need to use it in macro language.

The macro language equivalent is `curve()`.

The Curve Editor

The Curve editor belongs to a type of editor known as *family editor* which allows related sets of input specifications to be grouped in the same icon editor, each in its own tabbed pane (like the Simple Formula icon - see Figure III-3 on page III-71). The available working input panes are :

Curve - Provides input for plotting of y values against x values, either as a curve or scatter plot (depending on what you specify in the plotting attributes)

Bar - Provides input for plotting ranges of y values against x values as a set of vertical bars

Area - Provides input for plotting two series of x, y values, the first and last value of which are linked to form a closed area (which users usually plot as a filled polygon). As it is the area between the two series that is filled, it is usually best to make either the x or the y values similar between the two series.

Graph - Provides input for plotting a number of different curves (line, bar and area) in the same axis system. Each curve (and its plotting attributes) is specified by means of other Curve icons.

X Values (Curve and Bar Only)

Specifies the values of the x axis series of values. Elements are separated with a slash (/) character.

Y Values (Curve Only)

Specifies the values of the y axis series of values

High Y Values (Bar Only)

Specifies the high values of the y axis series of values. The top of the bar will be drawn at these values

Low Y Values (Bar Only)

Specifies the low values of the y axis series of values. The bottom of the bar will be drawn at these values or at the x axis depending on the value you have specified for **Axis Min Value** in the Axis icon entered in **Vertical Axis** (see below)

X1 Values (Area Only)

Specifies the x values of the first series of values

Y1 Values (Area Only)

Specifies the y values of the first series of values

X2 Values (Area Only)

Specifies the x values of the second series of values

Y2 Values (Area Only)

Specifies the x values of the second series of values. The first point and the last point of the series 1 and 2 above are plotted linked to form an area. The icon supplied to Attributes below determines how this area is plotted (not filled, filled, patterns, colours).

Curves (Graph Only)

Specifies the curves to be plotted together. Each curve is itself specified by a Curve icon (one could say the Curve icon is a recursive icon).

Horizontal Axis

Specifies the characteristics of the horizontal axis to be used in the plotting of the data. The icon field accepts only icons of icon Axis - see description in "Axis" on page III- 225. In the Axis icon you provide to the Graph input pane, be sure to specify values for **Axis Min Value** and **Axis Max Value** which cover the full range of values spanned by all the curves you want to plot

Vertical Axis

Specifies the characteristics of the vertical axis to be used in the plotting of the data. The icon field accepts only icons of icon Axis - see description in "Axis" on page III- 225. In the Axis icon you provide to the Graph input pane, be sure to specify values for **Axis Min Value** and **Axis Max Value** which cover the full range of values spanned by all the curves you want to plot

Title

Specifies the title of the plot - this will appear on the blue title bar of the plot window

Attributes

Specifies the attributes to be used in the plotting of the data (type and format). The icon field accepts only icons of icon Graph - see description in "Graph" on page III- 221. Not available in the Graph input pane, since each component curve will have its own attribute specification.

Actions on the Curve icon

The actions available for the Curve icon are *Visualise*, *Edit*, *Duplicate* and *Delete*

Visualise - this action carries out the visualisation of the specified data, using the default display window. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

BUDGET



This icon calculates and plots various budget quantities. It plots up to 4 budgets - Atmospheric Energy, Surface Energy, Hydrological, Top Radiation - using model budget fields as input.

The macro language equivalent is `budget ()`.

The Budget Editor

Points

Specifies the geographical scope of the budget calculations. Options are **Global** (global scope), **Land** (land surface points only) and **Sea** (ocean points only)

Data

Specifies the input data to be used. The assist button provides a tailor made MARS request in case you need some guidance in the data specification. It may be desirable to include `STEP=6` when the data is available (operational archive) in order to give a better representation of spin-up effects - this is part of the standard specification available from the icon help drawer. Steps of 6, 12, 24, 36 and so on in increments of 12, up to 720 (30 days) are all valid. If multiple dates are provided in the data then only the most recent date is used in the results; if budgets are to be averaged over a number of days then the simplest approach is to perform a separate data retrieval for each date, and provide their average to the Budget icon.

Land Sea Mask

Specifies the land-sea mask field. This must be specified if you specified **Land** or **Sea** for parameter **Points**. Use the icon help drawer for a ready made request.

Budget Type

Specifies the type of budget to be plotted. Options are **Atmospheric Energy**, **Surface Energy**, **Hydrological** and **Top Radiation**

Print Climate

Toggles printing of the climate values on the plot **On/Off**

Area

Specifies the coordinates of the area over which the specified budget is calculated. Enter coordinates (lat/long) of an area separated by a "/" (top left lat and long, bottom right lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42)

Vertical Axis

Specifies the characteristics of the vertical axis to be used in the plotting of the data. The icon field accepts only Axis icons - see description in "Axis" on page III- 225. Use the icon help drawer for a ready made request.

Title

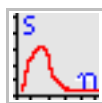
Specifies the title of the plot. Simply enter the required text.

Actions on the Budget Icon

The actions available for the Budget icon are *Visualise*, *Edit*, *Duplicate* and *Delete*

Visualise - this action visualises the derived output. If not visualised / executed before it also carries out the retrieval of the input data and derivation of the output. The data is left in a cache, so that subsequent actions happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

SPECTRA



This icon generates plots of spectra as a function of Legendre polynomial order.

The macro language equivalents are `spec_graph()` and `spec_contour()`.

The Spectra Editor

The Spectra editor belongs to a type of editor known as *family editor* which allows related sets of input specifications to be grouped in the same icon editor, each in its own tabbed pane (like the Simple Formula icon - see Figure III-3 on page III-71). The available working input panes are :

Spec Graph - To plot a graph of log/linear amplitude against log/linear n , where n is the Legendre polynomial order

Spec Contour - To plot a map contour plot of amplitude against wave number and forecast length

To operate with this icon first you need to select which output you want to produce by clicking the tab of the corresponding pane. Each input pane contains a different set of parameters, with some in common :

Data

Specifies the data required for the application. The assist button provides a tailor made MARS request in case you need some guidance in the data specification. Note that a curve is plotted for each value of `TIME` and `STEP`

Diurnal Cycle

Specifies whether to subtract the diurnal cycle. **Yes** to subtract, **No** otherwise

Truncation

Specifies the highest wave number in spectra plots

Title

Specifies a title of the spectra plot to be added to the automatically generated title. Just type in the required text

Vertical Axis

Specifies the characteristics of the vertical axis to be used in the plotting of the spectra. The icon field accepts only icons of icon Axis - see description in "Axis" on page III- 225

Horizontal Axis

Specifies the characteristics of the horizontal axis to be used in the plotting of the spectra. The icon field accepts only icons of icon Axis - see description in "Axis" on page III- 225. Only available for **Plot Type** set to **Spec Graph**

Smoothing

Specifies whether to smooth the plot curve(s). **Yes** to smooth, **No** otherwise. Only available for **Plot Type** set to **Spec Graph**

Tension

Specifies the smoothing interpolation. Only **Spline** is currently available. Only available for **Plot Type** set to **Spec Graph**

NI24int

Specifies whether to plot values only every 24 hours. **Yes** for only every 24 hours, **No** otherwise. Only available for **Plot Type** set to **Spec Contour**

Hilorad

Specifies the search radius for plotting of high/low markers. Only available for **Plot Type** set to **Spec Contour**

Actions on the Spectra Icon

The actions available for the Spectra icon are *Visualise*, *Edit*, *Duplicate* and *Delete*

Visualise - this action visualises the derived output. If not visualised / executed before it also carries out the retrieval of the input data and derivation of the output. The data is left in a cache, so that subsequent actions happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

SCORES



This icon generates and plots verification scores, specifically the rms or the correlation between fieldsets of analysis and forecast as a function of the forecast day

The macro language equivalent is `scores()`.

The Scores Editor

Score

Specifies the type of score to plot. Currently the only available options are **Rms** and **Correlation**.

Field 1

Specifies the first field used in the calculation of the specified score

Field 2

Specifies the second field used in the calculation of the specified score

Area

Specifies the coordinates of the area over which the specified score is calculated. Enter coordinates (lat/long) of an area separated by a "/" (top left lat and long, bottom right lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42

Actions on the Scores Icon

The actions available for the Scores icon are *Visualise*, *Edit*, *Duplicate* and *Delete*

Visualise - this action visualises the derived output. If not visualised / executed before it also carries out the retrieval of the input data and derivation of the output. The data is left in a cache, so that subsequent actions happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

TRAJCOMP



This icon generates modelled trajectory data. A related icon, TrajPlot-x.x (see page III-135), is used to visualise the resulting data. Note that this module is provided “as is” and has not been tested for accuracy in recent years.

The macro language equivalent is `trajectory()`.

TrajComp Default Input Data

The trajectory program uses as input data model level fields of U, V, W and LNSP. The program expects a single file containing these fields in no particular sequence.

A file containing this data (and also VO/D) is produced routinely. This file, stored on ecgate, is called `trajdata` and resides in the directory `/emos_data/trajectory`.

The contents of the file are as follows :

- Analysis fields of LNSP/U/V/W/VO/D for 15 days preceding (and excluding) the current day, followed by :
- Forecast fields produced the day before and verifying every 6 hours up to 10 days ahead

So, you can cover a period of 25 days in your input specification, from yesterday minus 15 days to yesterday plus 10 days. Results for yesterday and before will be derived from the analysis, results from today onwards are based on forecasts.

TrajComp Custom Input Data

If the data required by your input parameters is not fully included within the window around the present date described above, you will have to prepare the input data file yourself.

This is simply done by retrieving the data from MARS and storing it in a temporary location. Note that the data can be quite large - the default file is approximately 500MB.

An example Metview macro to build the trajectory file is presented below. This example requests only analysis data; to request forecast data amend the required parameters. Remember that data from an additional retrieval may be added to the original data using the `append()` function.

```

dest_file = "/tmp/xxx/traj/trajdata"

trajdata = retrieve
(
  expver      : 01,
  levtype     : "ml",
  levelist    : [1, "to", 60, "by", 1],
  param       : ["u", "v", "w", "Insp"],
  date        : [20030701, "to", 20030705],
  time        : [0, "to", 18, "by", 6],
  step        : 0,
  resol       : 106
)

write (dest_file, trajdata)

```

Remember that if you use a data file other than the default one routinely produced for the current period, this must be specified in the TrajComp Editor.

The TrajComp Editor

Date

Specifies the day of the trajectory start or end - see **Timestep In Mins**. The default value is -1 (yesterday), but you can use other formats, such as YYMMDD or YYYY-MM-DD.

Hour

The hour (at ECMWF this should be 00, 06, 12 or 18) of the trajectory start or end - see **Timestep In Mins**.

Length In Days

The length, in days, of the trajectory calculation.

Timestep in Mins

Time step for the calculations, in minutes. A positive value indicates that the trajectory will be calculated forwards in time from the **Date/Hour**. A negative values indicates that the trajectory will be calculated backwards in time from the **Date/Hour**.

Parcel Latitudes

A list of latitudes, one for each parcel whose trajectory is to be calculated. List items are separated with a slash (/) character. The number of list elements in **Parcel Latitudes** and **Parcel Longitudes** must be the same. If calculating for just one geographical point, then only one latitude and longitude is needed, no matter how many **Parcel Levels** are specified. If calculating for more than one geographical point, then the number of latitudes and longitudes must match the number of **Parcel Levels**.

Parcel Longitudes

A list of longitudes, one for each parcel whose trajectory is to be calculated. List items are separated with a slash (/) character.

Parcel Levels

A list of initial levels, one for each parcel whose trajectory is to be calculated. List items are separated with a slash (/) character.

Logfile Path

Path to a logfile where output from the trajectory module will be written.

Database

Determines whether the database containing the source data for the trajectory calculations is in a predefined location (**Standard**) or a **Custom Location**. The predefined location is set to /
emos_data/trajectory/trajdata.

Database Path

Only available if **Database** is set to **Custom Location**.

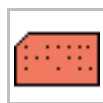
Actions on the TrajComp Icon

The actions available for the TrajComp icon are *Execute*, *Edit*, *Duplicate* and *Delete*.

Execute - this action calculates the trajectory output. The data is put in a cache, so that subsequent actions happen much faster.

Save - this action allows you to save the text file that contains the output from this icon. This text file can then be used as direct input to the TrajPlot-x.x icon.

TRAJPLOT-X.X



This icon plots the output of a TrajComp icon (see page III-131). It is a Metview Macro that creates a user interface from which the user can specify how to plot the data. The icon name carries the version number of the macro at the end (described here as 'x.x'). Each point on the trajectory is marked with a symbol. At user-defined intervals, the date and level are also indicated.

In order to use the icon, *Execute* it. A user interface is displayed, as described below. Clicking *Apply* runs the macro and plots the data.

The TrajPlot-x.x Editor

Trajectory Output

Specifies the output from a trajectory calculation. Drop a TrajPlot icon or its output text file into this field.

Symbol Descriptor

Determines the symbol to use to mark the trajectory points. For a table of numbers/symbols, see the MAGICS Users Guide, Chapter XII - Symbol Plotting.

Symbol Height

Determines the height of the markers, in cm.

Symbol Colour

Determines the colour of the trajectory markers.

Level And Date Height

Determines the height of the date and level text, in cm.

Level And Date Frequency

Determines the plotting frequency of the level and date indicators. For example, a frequency of 6 means that the level and date will only be plotted for every sixth point.

Date Colour

Determines the colour of the date text.

Level Colour

Determines the colour of the level text.

Plotting Option

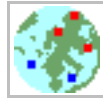
If multiple trajectories are specified in the input TrajPlot icon, they can be plotted either **All In One Frame**, or **One Per Frame**.

Actions on the TrajPlot-x.x Icon

The actions available for the TrajComp icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate* and *Delete*.

Execute, *Visualise*, *Examine*, *Save* - these all have the effect of displaying the user interface, thereby allowing the user to plot the input trajectories.

DATA COVERAGE (ECMWF ONLY)



This icon takes BUFR data as input and processes them by grouping the observations according to a variety of user defined quality related criteria. The user can either plot the result or save it to a geoints file. You may find more material on observations in the manual pages for the Observation Filter icon (see "Observation Filter" on page III- 53)

The macro language equivalent is `datacoverage()`.

Brief Overview

The objective of this icon is to provide ways to assess the observations arriving at the ECMWF. You can check for :

- Quality of the observations - You can plot the quality control values (for conventional data - for satellite data you get the satellite identifier) of the observations or of a specified meteorological parameter (data descriptor)
- Arrival time of the observations - You can check which observations arrived later than a given time - if this cutoff corresponds to the latest useful time of arrival of an observation (analysis cutoff time) you can identify the stations providing these late observations

If you do not specify an arrival cutoff time you only plot the quality control value. Specifying an arrival time cutoff, plots the quality control value for the observations arriving on time and flags the late arrivals with a symbol.

The Data Coverage Editor

Data

Specifies the input data. The assist button provides a tailor made MARS request in case you need some guidance in the data specification. The input data must be observations - hence supply a MARS request retrieving observations or a BUFR icon

Checking Date

Specifies a reference date to be used in the observation checking

Checking Time

Specifies a reference time to be used in the observation checking. This should be one of the standard observation times when observations are actually taken. The departure from this time defining whether an observation is late or not, is specified in **Cutoff Time in Minutes**

Cutoff Time in Minutes

Specifies the offset period from the checking date/time. Choose a value with the slider - the scale is in minutes. This value defines the tolerance used to define whether an observation is late or not. If you specify 0, the late / on time status of the observations is not plotted

Plot Subsets

Toggles the plotting of subsets **On/Off**. If **On**, all subsets in a multisubset BUFR message will be processed. For some observation data (some of the satellite data) this is terribly slow, so you can switch this **Off**. This means that only the first subset in a multisubset message will be processed (for satellite data it can still be useful to see the orbits etc)

Descriptors to Check

Specifies for which parameters (meteorological quantities) the quality control values are checked. To specify the parameter enter its unique descriptor value (a numerical code). The descriptor value is of the form *xxYYZ*, where *xx* defines the class (e.g. 12 = Temperature class) and *YYZ* the parameter within that class (e.g. 12004 = Dry bulb Temperature at 2m). If you do not know the descriptor value for the required parameter, you have to look it up in the "BUFR User Guide and Reference Manual" (ECMWF Meteorological Bulletin M1.4/4) - see "BUFR Table B", pages 99-108. The significance of code and flag values for non-quantitative parameters are given in the same reference, in "BUFR code table", pages 111-154. For the most common parameters, it may be quicker to launch an Observation Filter icon editor window and consult the assist button of its input parameter **Parameter** (see "Observation Filter" on page III- 53)

Symbol

Specifies the characteristics (colours and types) of the symbols used in the plotting of the processed observations. The icon field accepts only icons of icon Symbol Plot- see description in "Annotation" on page III- 215. It is recommended that you use the tailor made Symbol icon provided by the assist button. If the plot requires more symbol entries than are present in the symbol icon, then the system defaults will be used for those missing entries. User-defined entries will always be used in preference to the defaults.

Actions on the Data Coverage Icon

All actions from the icon menu are available for the Data Coverage icon. The ones requiring some explanation are :

Execute - this action carries out the retrieval of the input data from the archives. The retrieved data is stored in a cache on your workstation, but it is not visualised. Apart from the changes of colour to the icon name, nothing else visible happens.

Visualise - this action carries out the visualisation of the data, using a default display window. If the data has not been retrieved before, the visualise action also carries out the retrieval of the data into the cache. If the data is already in the cache, the visualisation happens much faster. The plotting of observations quality control value and/or late/on time status uses symbols and you will obtain a descriptive legend in the plot.

Save - this action allows you to save the filtered data as a BUFR file. It works in the same way as for a MARS icon, in that an I/O window pops up where you specify the path and name for the destination file. The only difference is that a BUFR icon is created rather than a GRIB icon : see "Saving MARS Retrieval Output (as Files)" on page III- 30, for details of operation of the I/O window

AVERAGE VIEW



This icon is a plotting specification for average (zonal or meridional) cross-section plots. It specifies the averaging area and the axis details as well as the plot positioning in the plot frame of the display window / paper sheet and the overlay of different data units in the same plot.

This icon, when incorporated in a display window layout allows you to obtain an average cross-section plot directly from a suitable (multi-level) data unit without resorting to the use of the Average application icon (see "Average Data" on page III- 87). Hence you will find a few input parameters common to both icons.

To understand how this works please see "The View Concept in Visualisation" on page I- 65. Here and in the chapter "Views and Visualisation" on page I- 86 you can find further details on the role and usage of view icons in the visualisation process.

The macro language equivalent is `averageview()`.

The Average View Editor

Area

Specifies the coordinates of the area over which the average profile is calculated. Enter coordinates (lat/long) of an area separated by a "/" (top left lat and long, bottom right lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42).

Direction

Specifies the direction along which the averaging of the variable is performed. Options are **North South** and **East West**. For **North South**, the averaging is weighted by $\cos(\text{latitude})$

Bottom Pressure

Specifies the lower limit of the cross section, in hPa

Top Pressure

Specifies the upper limit of the cross section, in hPa

Pressure Level Axis

Specifies the type of pressure axis - **Linear** or **Logarithmic**

Overlay Control

Specifies details of the overlaying of data units in the same plot. Drop an Overlay Control icon - see "Overlay Control" on page III- 237.

Subpage X Position

Specifies the X offset of the plot away from the left side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Subpage Y Position

Specifies the Y offset of the plot away from the bottom side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Subpage X Length

Specifies the X length of the plot counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Subpage Y Length

As above but for the Y length of the plot.

Page Frame

Toggles the plotting of a border line around the plot frame **On/Off**.

Page Frame Parameters

Comprising **Page Frame Colour**, **Page Frame Line Style** and **Page Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Page Frame** set to **On**.

Page Id Line

Toggles the plotting of plot identification line **On/Off**.

Page Id Line User Text

Specifies user text to be added to the plot identification line. Only available for **Page Id Line** set to **On**.

Subpage Frame

Toggles the plotting of a border line around the plot itself **On/Off**. In most cases you will want this to be left **On**. When **Off** the sides of the plot not equipped with axis will not be plotted.

Subpage Frame Parameters

Comprising **Subpage Frame Colour**, **Subpage Frame Line Style** and **Subpage Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Subpage Frame** set to **On**.

Subpage Background Colour

Specifies the colour of the background of the plot (i.e. not affected by visual definitions like contour shadings or lines).

Actions on the Average View Icon

The actions available for the Average View icon are *Execute*, *Visualise*, *Edit*, *Duplicate* and *Delete*.

Execute - this action has the same result as the *Visualise* action.

Visualise - this action results in a visualisation of an empty Average plot with default data settings. This is useful to check the defaults or to start visualisation work from here.

CROSS SECTION VIEW



This icon is a plotting specification for cross-section plots. It specifies the coordinates of the transect line and the axis details as well as the plot positioning in the plot frame of the display window / paper sheet and the overlay of different data units in the same plot.

This icon, when incorporated in a display window layout allows you to obtain a cross-section plot directly from a suitable (multi-level) data unit without resorting to the use of the Cross-Section application icon (see "Cross-Section Data" on page III- 89). Hence you will find a few input parameters common to both icons.

To understand how this works please see "The View Concept in Visualisation" on page I- 65. Here and in the chapter "Views and Visualisation" on page I- 86 you can find further details on the role and usage of view icons in the visualisation process.

The macro language equivalent is `xsectview()`

The Cross-Section View Editor

Line

Specifies the coordinates of a transect line along which the cross-section is calculated. Enter coordinates (lat/long) of a line separated by a "/" (easternmost lat and long, westernmost lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42).

Bottom Pressure

Specifies the lower limit of the cross section, in hPa

Top Pressure

Specifies the upper limit of the cross section, in hPa

Wind Parallel

Toggles the projection of the horizontal wind onto the cross section plane **On/Off**. Only this projection will be plotted using wind arrows

Wind Perpendicular

Toggles the projection of the horizontal wind component onto a plane perpendicular to the cross section plane **On/Off**. Only this projection will be plotted using contour lines

Wind Intensity

Toggles plotting of the intensity of the wind **On/Off**. If one of the previous two parameters is **On**, then the intensity is that of the specified projection. Wind intensity plots are plotted with contours

Pressure Level Axis

Specifies the type of pressure axis - **Linear** or **Logarithmic**

Overlay Control

Specifies details of the overlaying of data units in the same plot. Drop an Overlay Control icon - see "Overlay Control" on page III- 237.

Subpage X Position

Specifies the X offset of the plot away from the left side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Subpage Y Position

Specifies the Y offset of the plot away from the bottom side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Subpage X Length

Specifies the X length of the plot counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Subpage Y Length

As above but for the Y length of the plot.

Page Frame

Toggles the plotting of a border line around the plot frame **On/Off**.

Page Frame Parameters

Comprising **Page Frame Colour**, **Page Frame Line Style** and **Page Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Page Frame** set to **On**.

Page Id Line

Toggles the plotting of plot identification line **On/Off**.

Page Id Line User Text

Specifies user text to be added to the plot identification line. Only available for **Page Id Line** set to **On**.

Subpage Frame

Toggles the plotting of a border line around the plot itself **On/Off**. In most cases you will want this to be left **On**. When **Off** the sides of the plot not equipped with axis will not be plotted.

Subpage Frame Parameters

Comprising **Subpage Frame Colour**, **Subpage Frame Line Style** and **Subpage Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Subpage Frame** set to **On**.

Subpage Background Colour

Specifies the colour of the background of the plot (i.e. not affected by visual definitions like contour shadings or lines).

Actions on the Cross Section View Icon

The actions available for the Cross Section View icon are *Execute*, *Visualise*, *Edit*, *Duplicate* and *Delete*.

Execute - this action has the same result as the *Visualise* action.

Visualise - this action results in a visualisation of an empty Cross Section plot with default data settings. This is useful to check the defaults or to start visualisation work from here. Even better is to customise interactively the resulting view (e.g. changing its geographic characteristics) and then save the results back to the icon where you started from.

HOVMØLLER VIEW



This icon is a plotting specification for Hovmøller diagram plots. It specifies the plot positioning in the plot frame of the display window / paper sheet and the overlay of different data units in the same plot.

Note that to obtain a Hovmøller diagram you should specify your data units by means of a Hovmøller Data icon ("Hovmøller Data" on page III- 93).

To understand how this works please see "The View Concept in Visualisation" on page I- 65. Here and in the chapter "Views and Visualisation" on page I- 86 you can find further details on the role and usage of view icons in the visualisation process.

The macro language equivalent is `hovmoellerview()`.

The Hovmøller View Editor

Overlay Control

Specifies details of the overlaying of data units in the same plot. Drop an Overlay Control icon - see "Overlay Control" on page III- 237.

Subpage X Position

Specifies the X offset of the plot away from the left side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Subpage Y Position

Specifies the Y offset of the plot away from the bottom side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Subpage X Length

Specifies the X length of the plot counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Subpage Y Length

As above but for the Y length of the plot.

Page Frame

Toggles the plotting of a border line around the plot frame **On/Off**.

Page Frame Parameters

Comprising **Page Frame Colour**, **Page Frame Line Style** and **Page Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Page Frame** set to **On**.

Page Id Line

Toggles the plotting of plot identification line **On/Off**.

Page Id Line User Text

Specifies user text to be added to the plot identification line. Only available for **Page Id Line** set to **On**.

Subpage Frame

Toggles the plotting of a border line around the plot itself **On/Off**. In most cases you will want this to be left **On**. When **Off** the sides of the plot not equipped with axis will not be plotted.

Subpage Frame Parameters

Comprising **Subpage Frame Colour**, **Subpage Frame Line Style** and **Subpage Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Subpage Frame** set to **On**.

Subpage Background Colour

Specifies the colour of the background of the plot (i.e. not affected by visual definitions like contour shadings or lines).

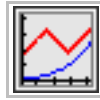
Actions on the Hovmøller View Icon

The actions available for the Hovmøller View icon are *Execute*, *Visualise*, *Edit*, *Duplicate* and *Delete*.

Execute - this action has the same result as the *Visualise* action.

Visualise - this action results in a visualisation of an empty Hovmøller diagram plot with default data settings. This is useful to check the defaults or to start visualisation work from here. You can also interactively customise the visualised view and then save the results back to the icon.

CURVE VIEW



This icon is a plotting specification for curve plots. It specifies the axis details as well as the plot positioning in the plot frame of the display window / paper sheet and the overlay of different data units in the same plot.

The macro language equivalent is `curveview()`.

The Curve View Editor

Horizontal Axis

Specifies the characteristics of the horizontal axis to be used in the plotting of the curves. The icon field accepts only icons of icon Axis - see description in "Axis" on page III- 225

Vertical Axis

Specifies the characteristics of the vertical axis to be used in the plotting of the curves. The icon field accepts only icons of icon Axis - see description in "Axis" on page III- 225

Overlay Control

Specifies details of the overlaying of different curves in the same plot. Drop an Overlay Control icon - see "Overlay Control" on page III- 237.

Subpage X Position

Specifies the X offset of the plot away from the left side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Subpage Y Position

Specifies the Y offset of the plot away from the bottom side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Subpage X Length

Specifies the X length of the plot counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Subpage Y Length

As above but for the Y length of the plot.

Page Frame

Toggles the plotting of a border line around the plot frame **On/Off**.

Page Frame Parameters

Comprising **Page Frame Colour**, **Page Frame Line Style** and **Page Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Page Frame** set to **On**.

Page Id Line

Toggles the plotting of plot identification line **On/Off**.

Page Id Line User Text

Specifies user text to be added to the plot identification line. Only available for **Page Id Line** set to **On**.

Subpage Frame

Toggles the plotting of a border line around the plot itself **On/Off**. In most cases you will want this to be left **On**. When **Off** the sides of the plot not equipped with axis will not be plotted.

Subpage Frame Parameters

Comprising **Subpage Frame Colour**, **Subpage Frame Line Style** and **Subpage Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Subpage Frame** set to **On**.

Subpage Background Colour

Specifies the colour of the background of the plot (i.e. not affected by visual definitions like contour shadings or lines).

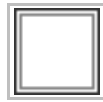
Actions on the Curve View Icon

The actions available for the Curve View icon are *Execute*, *Visualise*, *Edit*, *Duplicate* and *Delete*

Execute - this action has the same result as the *Visualise* action.

Visualise - this action results in a visualisation of an empty Curve plot with default data settings. This is useful to check the defaults or to start visualisation work from here. Even better is to customise interactively the resulting view and then save the results back to the icon where you started from. Note that the usual tools for changing the geography are not available for a Curve View.

EMPTY VIEW



This icon is used for layout purposes. See "Overview of Layout Design" on page I- 71 for details.

The macro language equivalent is `emptyview()`.

The Empty View Editor

Subpage X Position

Specifies the X offset of the plot away from the left side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Subpage Y Position

Specifies the Y offset of the plot away from the bottom side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Subpage X Length

Specifies the X length of the plot counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Subpage Y Length

As above but for the Y length of the plot.

Page Frame

Toggles the plotting of a border line around the plot frame **On/Off**.

Page Frame Parameters

Comprising **Page Frame Colour**, **Page Frame Line Style** and **Page Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Page Frame** set to **On**.

Page Id Line

Toggles the plotting of plot identification line **On/Off**.

Page Id Line User Text

Specifies user text to be added to the plot identification line. Only available for **Page Id Line** set to **On**.

Subpage Frame

Toggles the plotting of a border line around the plot itself **On/Off**. In most cases you will want this to be left **On**. When **Off** the sides of the plot not equipped with axis will not be plotted.

Subpage Frame Parameters

Comprising **Subpage Frame Colour**, **Subpage Frame Line Style** and **Subpage Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Subpage Frame** set to **On**.

Subpage Background Colour

Specifies the colour of the background of the plot (i.e. not affected by visual definitions like contour shadings or lines).

Actions on the Empty View Icon

The actions available for the Empty View icon are *Edit*, *Duplicate* and *Delete*

IMPORT VIEW



This icon is a plotting specification for imported graphics files (PS, JPEG, PNG formats). It specifies the graphics file positioning in the plot frame of the display window / paper sheet.

The macro language equivalent is `importview()`.

The Import View Editor

Subpage X Position

Specifies the X offset of the graphics file away from the left side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Subpage Y Position

Specifies the Y offset of the graphics file away from the bottom side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Subpage X Length

Specifies the X length of the graphics file counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Subpage Y Length

As above but for the Y length of the graphics file.

Page Frame

Toggles the plotting of a border line around the plot frame **On/Off**.

Page Frame Parameters

Comprising **Page Frame Colour**, **Page Frame Line Style** and **Page Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Page Frame** set to **On**.

Page Id Line

Toggles the plotting of plot identification line **On/Off**.

Page Id Line User Text

Specifies user text to be added to the plot identification line. Only available for **Page Id Line** set to **On**.

Subpage Frame

Toggles the plotting of a border line around the plot itself **On/Off**. In most cases you will want this to be left **On**. When **Off** the sides of the plot not equipped with axis will not be plotted.

Subpage Frame Parameters

Comprising **Subpage Frame Colour**, **Subpage Frame Line Style** and **Subpage Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Subpage Frame** set to **On**.

Subpage Background Colour

Specifies the colour of the background of the plot (i.e. not affected by visual definitions like contour shadings or lines).

Actions on the Import View Icon

The actions available for the Import View icon are *Execute*, *Visualise*, *Edit*, *Duplicate* and *Delete*

Execute - this action has the same result as the *Visualise* action.

Visualise - this action results in a visualisation of an empty Import plot with default data settings. This is useful to check the defaults or to start visualisation work from here. Even better is to customise interactively the resulting view and then save the results back to the icon where you started from.

MAP VIEW



This icon is a plotting specification for 2D geographical (map) plots. It specifies the area displayed, the projection to be used for the data, the characteristics of the map display (e.g. grid lines and spacing), the plot positioning in the plot frame of the display window / paper sheet and the overlay of different data units in the same plot.

To understand how this works please see "The View Concept in Visualisation" on page I- 65. Here and in the chapter "Views and Visualisation" on page I- 86 you can find further details on the role and usage of view icons in the visualisation process.

The macro language equivalent is `mapview()`.

The Map View Editor

Coastlines

Specifies the characteristics of the coastlines to be used in the plot. Drop a Coastlines icon - see "Coastlines" on page III- 207. You can use one of those contained in the icon assist drawer.

Map Projection

Specifies the projection to be used in the plotting of the data. Note that the map projection parameters (**Map Projection**, **Map Hemisphere**, **Area** and **Map Vertical Longitude**) are closely connected and must provide a consistent and valid set of coordinates. This is not checked within the Map View Editor, but when the view is used, the parameters are verified and if there is an inconsistency then default values will be substituted and a warning message issued in the Messages window (see "The Message Window" on page I- 126).

Map Hemisphere

Specifies the hemisphere to be plotted when selecting **Polar Stereographic** for the **Map Projection**.

Area

Specifies the coordinates of the area over which the average profile is calculated. Enter coordinates (lat/long) of an area separated by a "/" (South West lat and long, North East lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42).

Map Vertical Longitude

Specifies the vertical longitude for the projections. This is the longitude at the centre of the plot - vary this value to rotate or translate the map (depending on projection).

Map Parameters

This is a set of parameters only available for **Map Projection** set to **Lambert** - they include **Map Centre Latitude**, **Map Centre Longitude**, **Map Scale**, **Map Standard Latitude 1**, **Map Standard Latitude 2**. Enter suitable values for the required area of the plot.

Overlay Control

Specifies details of the overlaying of data units in the same plot. Drop an Overlay Control icon - see "Overlay Control" on page III- 237.

Subpage X Position

Specifies the X offset of the plot away from the left side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Subpage Y Position

Specifies the Y offset of the plot away from the bottom side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Subpage X Length

Specifies the X length of the plot counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Subpage Y Length

As above but for the Y length of the plot.

Page Frame

Toggles the plotting of a border line around the plot frame **On/Off**.

Page Frame Parameters

Comprising **Page Frame Colour**, **Page Frame Line Style** and **Page Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Page Frame** set to **On**.

Page Id Line

Toggles the plotting of plot identification line **On/Off**.

Page Id Line User Text

Specifies user text to be added to the plot identification line. Only available for **Page Id Line** set to **On**.

Subpage Frame

Toggles the plotting of a border line around the plot itself **On/Off**. In most cases you will want this to be left **On**. When **Off** the sides of the plot not equipped with axis will not be plotted.

Subpage Frame Parameters

Comprising **Subpage Frame Colour**, **Subpage Frame Line Style** and **Subpage Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Subpage Frame** set to **On**.

Subpage Background Colour

Specifies the colour of the background of the plot (i.e. not affected by visual definitions like contour shadings or lines).

Actions on the Map View Icon

The actions available for the Map View icon are *Execute*, *Visualise*, *Edit*, *Duplicate* and *Delete*

Execute - this action has the same result as the *Visualise* action.

Visualise - this action results in a visualisation of an empty Map plot with default data settings. This is useful to check the defaults or to start visualisation work from here.

SATELLITE VIEW



This icon is a plotting specification for satellite data. It specifies the satellite details, coastline information, the plot positioning in the plot frame of the display window / paper sheet and the overlay of different data units in the same plot.

To understand how this works please see "The View Concept in Visualisation" on page I- 65. Here and in the chapter "Views and Visualisation" on page I- 86 you can find further details on the role and usage of view icons in the visualisation process.

The macro language equivalent is `satelliteview()`.

The Satellite View Editor

Coastlines

Specifies the characteristics of the coastlines to be used in the plot. Drop a Coastlines icon - see "Coastlines" on page III- 207. You can use one of those contained in the icon assist drawer.

Subpage Map Sub Sat Longitude

Longitude (in thousandths of degrees) of point directly beneath satellite. Although satellite image data may only be displayed on a projection defined by its GRIB data, it is possible to select a satellite projection for any longitude position for the purposes of plotting contours, wind etc.

Image Subarea Selection

When **off**, the full satellite projection is displayed. When **on**, a rectangular subarea of the projection may be displayed by specifying its centre and one of its corners in the four parameters below.

Subpage Map Centre Latitude

Latitude of the centre of the subarea to be displayed.

Subpage Map Centre Longitude

Longitude of the centre of the subarea to be displayed.

Subpage Map Corner Latitude

Latitude of one of the corners of the subarea to be displayed. The corner could, for instance, be the bottom left hand or top right hand corner.

Subpage Map Corner Longitude

Longitude of one of the corners of the subarea to be displayed. The corner could, for instance, be the bottom left hand or top right hand corner.

Input Image Columns

Specifies the number of pixel columns in the satellite image.

Input Image Rows

Specifies the number of pixel rows in the satellite image.

Subpage Map Initial Column

Specifies the first column of the map that should be overlaid on the image data.

Subpage Map Initial Row

Specifies the first row of the map that should be overlaid on the image data.

Subpage Map Sub Sat X

Specifies the pixel column that lies directly underneath the satellite.

Subpage Map Sub Sat Y

Specifies the pixel row that lies directly underneath the satellite.

Subpage Map X Earth Diameter

The diameter of the Earth in pixels measured horizontally across the satellite image.

Subpage Map Y Earth Diameter

The diameter of the Earth in pixels measured vertically across the satellite image.

Subpage Map Grid Orientation

The orientation of the map grid in thousandths of degrees.

Subpage Map Camera Altitude

Specifies the altitude of the 'camera' looking down on the Earth. For satellite image data, this should be the satellite's altitude.

Overlay Control

Specifies details of the overlaying of data units in the same plot. Drop an Overlay Control icon - see "Overlay Control" on page III- 237.

Subpage X Position

Specifies the X offset of the plot away from the left side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Subpage Y Position

Specifies the Y offset of the plot away from the bottom side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Subpage X Length

Specifies the X length of the plot counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Subpage Y Length

As above but for the Y length of the plot.

Page Frame

Toggles the plotting of a border line around the plot frame **On/Off**.

Page Frame Parameters

Comprising **Page Frame Colour**, **Page Frame Line Style** and **Page Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Page Frame** set to **On**.

Page Id Line

Toggles the plotting of plot identification line **On/Off**.

Page Id Line User Text

Specifies user text to be added to the plot identification line. Only available for **Page Id Line** set to **On**.

Subpage Frame

Toggles the plotting of a border line around the plot itself **On/Off**. In most cases you will want this to be left **On**. When **Off** the sides of the plot not equipped with axis will not be plotted.

Subpage Frame Parameters

Comprising **Subpage Frame Colour**, **Subpage Frame Line Style** and **Subpage Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Subpage Frame** set to **On**.

Subpage Background Colour

Specifies the colour of the background of the plot (i.e. not affected by visual definitions like contour shadings or lines).

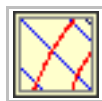
Actions on the Satellite View Icon

The actions available for the Satellite View icon are *Execute*, *Visualise*, *Edit*, *Duplicate* and *Delete*

Execute - this action has the same result as the *Visualise* action.

Visualise - this action results in a visualisation of an empty Satellite plot with default data settings. This is useful to check the defaults or to start visualisation work from here.

TEPHIGRAM VIEW



This icon is a plotting specification for tephigram plots. It specifies the tephigram axis limits, the plot positioning in the plot frame of the display window / paper sheet and the overlay of different data units in the same plot. Note that to obtain a tephigram you should specify your data units by means of a Tephigram Input icon ("Tephigram Data" on page III- 119).

To understand how this works please see "The View Concept in Visualisation" on page I- 65. Here and in the chapter "Views and Visualisation" on page I- 86 you can find further details on the role and usage of view icons in the visualisation process.

The macro language equivalent is `tephigramview()`.

The Tephigram View Editor

Minimum Temperature

Specifies the minimum value on the temperature axis of the tephigram plot.

Maximal Temperature

Specifies the maximum value on the temperature axis of the tephigram plot.

Bottom Pressure

Specifies the value at the bottom of the pressure axis of the tephigram plot.

Top Pressure

Specifies the value at the top of the pressure axis of the tephigram plot.

Background Thinning

Specifies that fewer background lines should be drawn in the plot.

Plot Wind

Toggles the plotting of the Wind Subframe on/off.

Overlay Control

Specifies details of the overlaying of data units in the same plot. Drop an Overlay Control icon - see "Overlay Control" on page III- 237.

Subpage X Position

Specifies the X offset of the plot away from the left side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Subpage Y Position

Specifies the Y offset of the plot away from the bottom side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Subpage X Length

Specifies the X length of the plot counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Subpage Y Length

As above but for the Y length of the plot.

Wind Subpage X Position

Specifies the X offset of the plot away from the left side of the wind plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Wind Subpage Y Position

Specifies the Y offset of the plot away from the bottom side of the wind plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Wind Subpage X Length

Specifies the X length of the plot counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the wind plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Wind Subpage Y Length

As above but for the Y length of the plot.

Page Frame

Toggles the plotting of a border line around the plot frame **On/Off**.

Page Frame Parameters

Comprising **Page Frame Colour**, **Page Frame Line Style** and **Page Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Page Frame** set to **On**.

Page Id Line

Toggles the plotting of plot identification line **On/Off**.

Page Id Line User Text

Specifies user text to be added to the plot identification line. Only available for **Page Id Line** set to **On**.

Subpage Frame

Toggles the plotting of a border line around the plot itself **On/Off**. In most cases you will want this to be left **On**. When **Off** the sides of the plot not equipped with axis will not be plotted.

Subpage Frame Parameters

Comprising **Subpage Frame Colour**, **Subpage Frame Line Style** and **Subpage Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Subpage Frame** set to **On**.

Subpage Background Colour

Specifies the colour of the background of the plot (i.e. not affected by visual definitions like contour shadings or lines).

Actions on the Tephigram View Icon

The actions available for the Tephigram View icon are *Execute*, *Visualise*, *Edit*, *Duplicate* and *Delete*

Execute - this action has the same result as the *Visualise* action.

Visualise - this action results in a visualisation of an empty Tephigram plot with default data settings. This is useful to check the defaults or to start visualisation work from here. Even better is to customise interactively the resulting view and then save the results back to the icon where you started from.

TEXT VIEW



This icon is a plotting specification for imported text. It specifies the text box positioning in the plot frame of the display window / paper sheet. This is used to provide plot frames that can be filled with user defined text (e.g. to provide comments on complex plots).

The user text is provided by means of Notes icons (see "Notes" on page III- 5) to be dropped in the plot frames containing this type of view. MAGIC3 text specifications must be used to control the text format (size, non-english characters, italics, bold, font types, etc,...) - see Chapter 14 ("Text Plotting") of the MAGIC3 Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>.

The macro language equivalent is `textview()`.

The Text View Editor

Subpage X Position

Specifies the X offset of the notes file away from the left side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Subpage Y Position

Specifies the Y offset of the notes file away from the bottom side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Subpage X Length

Specifies the X length of the notes file counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Subpage Y Length

As above but for the Y length of the notes file.

Page Frame

Toggles the plotting of a border line around the plot frame **On/Off**.

Page Frame Parameters

Comprising **Page Frame Colour**, **Page Frame Line Style** and **Page Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Page Frame** set to **On**.

Page Id Line

Toggles the plotting of plot identification line **On/Off**.

Page Id Line User Text

Specifies user text to be added to the plot identification line. Only available for **Page Id Line** set to **On**.

Subpage Frame

Toggles the plotting of a border line around the plot itself **On/Off**. In most cases you will want this to be left **On**. When **Off** the sides of the plot not equipped with axis will not be plotted.

Subpage Frame Parameters

Comprising **Subpage Frame Colour**, **Subpage Frame Line Style** and **Subpage Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Subpage Frame** set to **On**.

Subpage Background Colour

Specifies the colour of the background of the plot (i.e. not affected by visual definitions like contour shadings or lines).

Actions on the Text View Icon

The actions available for the Import View icon are *Execute*, *Visualise*, *Edit*, *Duplicate* and *Delete*

Execute - this action has the same result as the *Visualise* action.

Visualise - this action results in a visualisation of an empty Text plot with default data settings. This is useful to check the defaults or to start visualisation work from here.

VERTICAL PROFILEVIEW



This icon is a plotting specification for vertical profile plots. It specifies the location (point or area) and the axis details as well as the plot positioning in the plot frame of the display window / paper sheet and the overlay of different data units in the same plot.

This icon, when incorporated in a display window layout allows you to obtain a vertical profile plot directly from a suitable (multi-level) data unit without resorting to the use of the Vertical Profile application icon (see "Vertical Profile Data" on page III- 99). Hence, you will find a few input parameters common to both icons.

To understand how this works please see "The View Concept in Visualisation" on page I- 65. Here and in the chapter "Views and Visualisation" on page I- 86 you can find further details on the role and usage of view icons in the visualisation process.

The macro language equivalent is `verticalprofile()`.

The Vertical Profile View Editor

Input Mode

Specifies whether to derive the vertical profile for a **Point** or an **Area**

Point

Specifies the coordinates of the point for which the vertical profile is calculated. Enter coordinates (lat/long) of a point separated by a "/" ; alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42

Area

Specifies the coordinates of the area over which the averages composing the vertical profile are calculated. Enter coordinates (lat/long) of an area separated by a "/" (top left lat and long, bottom right lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42

Bottom Pressure

Specifies the lower limit of the vertical profile, in hPa

Top Pressure

Specifies the upper limit of the vertical profile, in hPa

Pressure Level Axis

Specifies the type of pressure axis - **Linear** or **Logarithmic**

Pressure Axis

Specifies the characteristics of the pressure (vertical) axis to be used in the plotting of the vertical profile. The icon field accepts only icons of icon Axis - see description in "Axis" on page III- 225

Value Axis

Specifies the characteristics of the horizontal (value) axis to be used in the plotting of the curves. The icon field accepts only icons of icon Axis - see description in "Axis" on page III- 225

Overlay Control

Specifies details of the overlaying of data units in the same plot. Drop an Overlay Control icon - see "Overlay Control" on page III- 237.

Subpage X Position

Specifies the X offset of the plot away from the left side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the X-dimension of the plot frame.

Subpage Y Position

Specifies the Y offset of the plot away from the bottom side of the plot frame (any subdivision of the display area - see "Overview of Layout Design" on page I- 71). This is expressed as a percentage of the Y-dimension of the plot frame.

Subpage X Length

Specifies the X length of the plot counting from the point defined by Subpage X Position. This is expressed as a percentage of the X-dimension of the plot frame. Hence the sum of this X length plus the X offset cannot exceed 100 (it is advised that it does not exceed 95 since you need some margin on the right for things like axis or map grid labels).

Subpage Y Length

As above but for the Y length of the plot.

Page Frame

Toggles the plotting of a border line around the plot frame **On/Off**.

Page Frame Parameters

Comprising **Page Frame Colour**, **Page Frame Line Style** and **Page Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Page Frame** set to **On**.

Page Id Line

Toggles the plotting of plot identification line **On/Off**.

Page Id Line User Text

Specifies user text to be added to the plot identification line. Only available for **Page Id Line** set to **On**.

Subpage Frame

Toggles the plotting of a border line around the plot itself **On/Off**. In most cases you will want this to be left **On**. When **Off** the sides of the plot not equipped with axis will not be plotted.

Subpage Frame Parameters

Comprising **Subpage Frame Colour**, **Subpage Frame Line Style** and **Subpage Frame Thickness**, these parameters specify the characteristics of the plot frame border line. Only available for **Subpage Frame** set to **On**.

Subpage Background Colour

Specifies the colour of the background of the plot (i.e. not affected by visual definitions like contour shadings or lines).

Actions on the Vertical Profile View Icon

The actions available for the Vertical Profile View icon are *Execute*, *Visualise*, *Edit*, *Duplicate* and *Delete*

Execute - this action has the same result as the *Visualise* action.

Visualise - this action results in a visualisation of an empty Vertical Profile plot with default data settings. This is useful to check the defaults or to start visualisation work from here. Even better is to customise interactively the resulting view (e.g. changing its geographic characteristics) and then save the results back to the icon where you started from.

GRIB TO VIS5D



This is the icon used for converting a Grib to Vis5D format. For more information on Vis5D, see the documentation at <http://www.ssec.wisc.edu/~billh/vis5d.html>. Note that the ECMWF is not responsible for information held on external sites.

The macro language equivalent is `grib_to_vis5d()`.

The Grib to Vis5D Editor

Data

Specifies the Grib data to be converted. Drop an icon that returns a Grib file, eg a Grib icon or a MARS request that returns Grib data.

Compression

This specifies the number of bytes used to store each grid point. Possible values are 1, 2 or 4.

Vis5D File

Specifies whether the resultant Vis5D file should be **Temporary** or **Permanent**.

Path

Only available if **Vis5D File** is set to **Permanent**. Specifies the path of the resultant Vis5D file.

If File Exists

Only available if **Vis5D File** is set to **Permanent**. Determines the outcome if a filename of the same name already exists at that location. Options are: **Fail** (the conversion fails), **Use It** (the original file is used and no conversion takes place), **Overwrite It** (the new file replaces the old one) and **Check Dates** (the old file is overwritten if it is older than its related files).

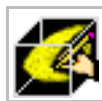
Actions on the Grib To Vis5D Icon

The actions available for the Grib To Vis5D icon are *Execute*, *Visualise*, *Examine*, *Save*, *Edit*, *Duplicate* and *Delete*.

Execute - this action performs the actual data conversion. if not visualised / executed before it also carries out the retrieval of the input data and derivation of the output. The data is left in a cache, so that subsequent actions happen much faster. For details on data visualisation see "Visualisation - an Overview" on page I- 62 of this Manual.

Visualise - if not visualised / executed before it also carries out the retrieval of the input data and derivation of the output. If a Vis5D file has been successfully created, then the Vis5D application will be launched in order to view the data.

Vis5D OBJECT



This icon defines a Vis5D plotting object (vertical slice, isosurface, etc.). To use this icon, drop it into a Vis5D window. Subsequent drops of different **Types** are cumulative additions to the display. For more information on Vis5D, see the documentation at <http://www.ssec.wisc.edu/~billh/vis5d.html>. Note that the ECMWF is not responsible for information held on external sites.

The macro language equivalent is `vis5d_object()`.

The Vis5D Object Editor

Type

Specifies the type of visualisation to be rendered. Options are: **Horizontal Slice**, **Vertical Slice**, **Isosurface**, **Coloured Horizontal Slice**, **Coloured Vertical Slice** and **Volume**.

Variable

Specifies the variable to be visualised.

Clone Variable

Specifies whether to make a clone of the variable. Clones are copies of existing variables. For example, you can use a cloned variable to make two different isosurfaces of the same variable simultaneously.

Attributes

Specifies whether to use default attributes or user-supplied attributes. Precisely which attribute parameters are available depends on which **Type** is selected.

Value

Specifies the value which is to be displayed.

Interval

Specifies the interval at which contours should be displayed.

Low

Specifies the lowest valued contour that should be displayed.

High

Specifies the highest valued contour that should be displayed.

Level

Specifies the grid level position of the slice.

Line

Specifies the coordinates of a transect line. Enter coordinates (lat/long) of a line separated by a "/" (easternmost lat and long, westernmost lat and long); alternatively, use the coordinate assist button (see "Geography help tool" in "Editor Window Help Tools" on page I- 42).

Colour

Specifies one of three colour modes. **Default** is the Vis5D default, **RGBA** allows you to explicitly define the RGBA components of the colour, and **Variable** means that the colour will depend on a data variable.

Red

The red component of the colour. Valid values range from 0.0 to 1.0.

Green

The green component of the colour. Valid values range from 0.0 to 1.0.

Blue

The blue component of the colour. Valid values range from 0.0 to 1.0.

Opacity

The opacity (alpha) component of the colour. Valid values range from 0.0 to 1.0. A value of 0.0 means the data will be completely transparent (invisible); and value of 1.0 means that the data will be completely opaque. Note that this parameter is not used for some types of data display.

Actions on the Vis5D Object Icon

The actions available for the Vis5D Object icon are *Edit*, *Duplicate* and *Delete*. You may drag the icon onto an active Vis5D window in order to apply its settings.

Vis5D SCRIPT



This icon allows you to write a Vis5D script. The Vis5D Script editor is essentially a simple text editor. For details on entering text into the editor, see "Working with Text Icon Editors" on page I- 51.

For more information on Vis5D, see the documentation at <http://www.ssec.wisc.edu/~billh/vis5d.html>. Note that the ECMWF is not responsible for information held on external sites.

VIS5D TRAJECTORY



To use this icon, drop it into a Vis5D window. For more information on Vis5D, see the documentation at <http://www.ssec.wisc.edu/~billh/vis5d.html>. Note that the ECMWF is not responsible for information held on external sites.

The macro language equivalent is `vis5d_trajectory()`.

The Vis5D Trajectory Editor

Set

Specifies the set you wish to define. There are 8 available sets.

Dates

The start and end dates for the trajectory calculation.

Position

Specifies the position used in the trajectory.

Replace Set

Set to **True** to replace the given set, or **False** to add to the given set.

Attributes

Determines whether the default attributes or user-provided attributes should be used.

Step

Specifies the time step for the trajectory.

Length

Trajectory length value. 1.0 is normal. Use a smaller value for shorter trajectories, a larger value for longer trajectories.

Ribbon

If **Yes**, then ribbons will be created. Otherwise, line segments will be created.

Colour

Specifies one of three colour modes. **Default** is the Vis5D default, **RGBA** allows you to explicitly define the RGBA components of the colour, and **Variable** means that the colour will depend on a data variable.

Red

The red component of the colour. Valid values range from 0.0 to 1.0.

Green

The green component of the colour. Valid values range from 0.0 to 1.0.

Blue

The blue component of the colour. Valid values range from 0.0 to 1.0.

Opacity

The opacity (alpha) component of the colour. Valid values range from 0.0 to 1.0. A value of 0.0 means the data will be completely transparent (invisible); and value of 1.0 means that the data will be completely opaque.

Colouring Variable

Determines which variable will be used in the colouring of the data visualisation. Only available if **Colour** is set to **Variable**.

Actions on the Vis5D Trajectory Icon

The actions available for the Vis5D Trajectory icon are *Edit*, *Duplicate* and *Delete*. You may drag the icon onto an active Vis5D window in order to apply its settings.

VIS5D WIND OBJECT



To use this icon, drop it into a Vis5D window. For more information on Vis5D, see the documentation at <http://www.ssec.wisc.edu/~billh/vis5d.html>. Note that the ECMWF is not responsible for information held on external sites.

The macro language equivalent is `vis5d_wind_object()`.

The Vis5D Wind Object Editor

Type

Specifies whether the type should be **Horizontal Wind**, **Vertical Wind**, **Horizontal Stream** or **Vertical Stream**.

Set

Specifies the set of wind slices to be defined. There are 2 available.

Attributes

Determines whether the default attributes or user-provided attributes should be used.

Scale

Wind vector scale length. The default value is 1.0.

Density

Density of vectors, where 1 is the default, 0.5 is half as many, etc.

Level

Grid level position of the slice.

Line

Position of the slice.

Colour

Specifies one of two colour modes. **Default** is the Vis5D default, while **RGBA** allows you to explicitly define the RGBA components of the colour.

Red

The red component of the colour. Valid values range from 0.0 to 1.0.

Green

The green component of the colour. Valid values range from 0.0 to 1.0.

Blue

The blue component of the colour. Valid values range from 0.0 to 1.0.

Opacity

The opacity (alpha) component of the colour. Valid values range from 0.0 to 1.0. A value of 0.0 means the data will be completely transparent (invisible); and value of 1.0 means that the data will be completely opaque.

Actions on the Vis5D Wind Object Icon

The actions available for the Vis5D Wind Object icon are *Edit*, *Duplicate* and *Delete*. You may drag the icon onto an active Vis5D window in order to apply its settings.

Vis5D WINDOW



To use this icon, either perform the **Execute** action on it, or drop it into a Vis5D window. For more information on Vis5D, see the documentation at <http://www.ssec.wisc.edu/~billh/vis5d.html>. Note that the ECMWF is not responsible for information held on external sites.

The macro language equivalent is `vis5d_window()`.

The Vis5D Window Editor

Area

Specifies the geographical area to be used.

Grid

Specifies the grid to use (number).

Levels

Specifies a list of grid levels.

Settings

The following settings can be left **As Is**, or set to **On**, **Off** or **Toggle** : **Box**, **Clock**, **Map**, **Topo**, **Perspective**, **Contour Numbers**, **Grid Coords**, **Pretty**, **Info**, **Probe**, **Cursor**, **Animrecord**, **Texture**, **Depthcue**, **Barbs**, **Julian**, **Snd Thta**, **Snd Thte**, **Snd W**, **Snd Ticks**, **Reverse**, **Alphamode**, **Hirestopo**, and **Samescale**.

Actions on the Vis5D Window Icon

The actions available for the Vis5D Wind Object icon are *Execute*, *Edit*, *Duplicate* and *Delete*. Executing the icon creates a new Vis5D window with the given settings. You may drag the icon onto an active Vis5D window in order to apply its settings.

CONTOUR



This is the visual definition used for plotting contours. It is the translation of the `MAGICS` contouring directives into a Metview icon.

The macro language equivalent is `pcont()`

Further details, not presented in this manual can be found in Chapter 5 ("Contouring") of the `MAGICS` Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>.

Main Contour Features

The Contour editor has a very large number of parameters. These are grouped according to their role and some groups are available/unavailable as a block. They are assigned sensible defaults, to enable you to produce good quality plots with minimum changes to the default values.

If you have a clear idea of what your contour should look like there should be no problem in getting oriented among the many `pcont()` commands.

There are decisions concerning the type of contouring you want - contour lines with no colour shading, colour shading with no contour lines, both contour lines and colour shading. Single contouring or 2 styles (split contour) applied to two ranges of values (e.g. positive and negative). What should be the contour spacing or the width of the colour shade bands? These basic decisions screen out large numbers of parameters as **On/Off** toggles make whole groups of parameters unavailable (or available as the case may be).

Having selected the basic options, follow with decisions concerning extra information - will you plot labels or not, will you plot the extrema (maxima and minima) or not? If you do, what symbol, format, frequency do you want. Will you highlight given contour values, e.g. every *n* lines? - this extra information is bundled into sets of parameters which become unavailable should they not be required (e.g. setting **Contour Label** or **Contour Highlight** to **Off**).

Finally, you will find parameters for further control of the contouring - choice of contouring methods, scaling of data, interpolation to a regular grid, screening out of field values. The scaling of data (**Grib Scaling** parameters in page III-193) may be the most relevant for your uses.

The Contour Editor

Legend

Toggles the user defined legend **On** or **Off**

Legend User Text

Type the text you want to go with the legend. Unavailable if **Legend** set to **Off**

Legend Entry

Drop a Legend Entry icon (see "Legend Entry" on page III- 231). Unavailable unless **Legend** is set to **On**

Contour Method

Specifies the contour method to be used - **C3** (default), **Linear** and **Conicon**. Note that **C3** and **Conicon** currently are the same (they used to refer to different versions of the CONICON method) :

- **Linear** - Direct contour threading method using linear interpolation. This is computationally cheaper but **Contour Shade...** parameters become unavailable as contour shading is not allowed.
- **Conicon** - Seamed quadratic element method. Allows contour shading

Contour Data Transformation

Options are **Normal** and **Isotachs** (contours of equal wind speed, hence only applicable to plotting of wind or any other vector fields, e.g. a Vector icon returning a vector from wave height and direction input).

Contour

Toggles contour lines **On** or **Off**. If using contour shading you may prefer not to show the contour lines separating different colour regions.

Contour Min Level

Only values equal to or above the specified value will be plotted. Enter a numerical value consistent with the values to be plotted

Contour Max Level

Only values equal to or below the specified value will be plotted. Enter a numerical value consistent with the values to be plotted

Contour Level Selection Type

This parameter specifies how the contour lines are assigned to the data to be plotted. Options are **Count**, **Interval** and **Level List** :

- **Count** - a fixed number of contour lines will be present in the plot (see **Contour Level Count**), plotted either side of the value specified in **Contour Reference Level**.
- **Interval** - contour lines are plotted at a regular interval (see **Contour Interval**) either side of the value specified in **Contour Reference Level**.
- **Level List** - contour lines are plotted for a specified list of values (see **Contour Level List**).

Contour Level Count

Specifies the number of contour lines to be present in the plot. Requires **Contour Level Selection Type** set to **Count**. MAGICS derives a contour interval from the specified number of lines and the range (max-min) of values in the field. This interval must be an integer number, so you may not obtain the exact number of contour lines you specify. The contour lines are plotted at this regular interval either side of the level specified in **Contour Reference Level**.

Contour Level List

Specifies a list of values at which the contour lines will be drawn. Requires **Contour Level Selection Type** set to **Level List**. Values are separated by forward slashes (n1/n2/ . . .); you must use this for irregularly spaced contour lines.

Contour Interval

Specifies the interval between two consecutive contour lines. Requires **Contour Level Selection Type** set to **Interval**. The contour lines are plotted at this interval either side of the level specified in **Contour Reference Level**.

Contour Missing Values Present

Specifies whether the field to be contoured has missing values (missing, not zero!)

Contour Reference Level

Specifies a value to be used as reference level. This value is always assigned a contour line and the other contour lines are plotted at a regular interval either side of this value. **Contour Level Selection Type** must be set to **Count** or **Interval** and the contour interval is defined via **Contour Level Count** or **Contour Interval**, respectively.

Contour Shade

Toggles the contour shading **On/Off** : the set of contour shading specification parameters that follow becomes available/unavailable

Contour Shade Technique

Specifies the technique for shading the area between two contour levels (known as a *shading band*). **Contour Method** must be set to **C3/Conicon**. Choices are **Polygon Shading**, **Cell Shading** and **Marker Shading** :

- **Polygon Shading** - Contours are formed into closed polygons and the areas within the polygons are shaded
- **Cell Shading** - Similarly to satellite imaging, shades every cell (pixel) according to the value of the field at that cell.
- **Marker Shading** - Plots markers on the grid points where the colour, height and type of marker depend on the value at the grid point. You need to specify parameters **Contour Shade Colour Table/Height Table/Marker Table** (see below). It is recommended to use **Contour Level Selection Type** set to **Level List** and to specify in **Contour Level List** a number of levels one above the number of elements in the three **Contour Shade Table** parameters.

Contour Shade Reduction Method

Specifies the reduction of number of points in polygons. It is only available if **Contour Shade Technique** is set to **Polygon Shading**. Some fields, particularly derived fields, have a complicated structure / very fine resolution which may make contouring by polygon shading CPU expensive.

MAGICS is able to reduce the number of points in the polygons to an acceptable level, but bear in mind that this may affect the quality and accuracy of the contouring. Reduction method options are :

- **Pre Calculation** - The reduction takes place before the shading process
- **Post Calculation** - The reduction takes place after the shading process (more expensive in CPU time)
- **None** - No reduction takes place (most expensive in CPU time)

Contour Shade Min Level

Only values equal to or above the specified value will be plotted. This should be set to a level that will actually be contoured.

Contour Shade Max Level

Only values equal to or below the specified value will be plotted. This should be set to a level that will actually be contoured.

Contour Shade Colour Method

Specifies which method to use for the determination of the colours to be used in the shading. Not available for **Contour Shade Technique** set to **Marker**. Options are **Calculate** and **List** :

- **Calculate** - the colour of the bands is calculated from the values in **Contour Shade Min Level**, **Contour Shade Max Level**, **Contour Shade Min Level Colour** and **Contour Shade Max Level Colour**. Colour shading is performed in such a way that starting with **Contour Shade Min Level Colour**, the colour of each subsequent band will vary from the preceding band by a factor that depends on the number of bands to be shaded. The colour of the last band is **Contour Shade Max Level Colour** and the shading of intermediate bands is calculated internally by modifying the hue, saturation and intensity. See also **Contour Shade Colour Direction** below.
- **List** - the user specifies a list of colours to be used

Contour Shade Min Level Colour

Specifies the colour assigned to **Contour Shade Min Level**. To use hues of a single colour select **WHITE** for this parameter and select the colour name in **Contour Shade Max Level Colour** (or vice-versa for the reverse sequence). Gradation to black is performed in the same way using **BLACK** instead of **WHITE**. Not available for **Contour Shade Technique** set to **Marker** or **Contour Shade Colour Method** set to **List**.

Contour Shade Max Level Colour

Specifies the colour assigned to **Contour Shade Max Level**. See previous parameter for details on colour hue gradation. Not available for **Contour Shade Technique** set to **Marker** or **Contour Shade Colour Method** set to **List**.

Contour Shade Colour Direction

Toggles the direction of colour selection in the 360° Hue scale between **Clockwise** and **Anti Clockwise**.

Contour Shade Colour List

Specifies a list of colours to be used in the shading. Not available for **Contour Shade Technique** set to **Marker**. **Contour Shade Colour Method** must be set to **List**. On the colour help tool, click the colours in the desired order of appearance. New colours are added to the end of the list, hence be careful with misplacement of colours in a long list. To remove a colour from the list, click left again on it.

Contour Shade Colour Table

Specifies a list of colours to be used with the selected markers. On the colour help tool, click the colours in the desired order of appearance. New colours are added to the end of the list, hence be careful with misplacement of colours in a long list. To remove a colour from the list, click left again on it. It is recommended that the number of elements in this list be one less than the number of levels to be used - see **Marker Shading** in **Contour Shading Technique** above. If only one value is set, it will apply to all shading bands

Contour Shade Height Table

Specifies a list of heights (in cm) to be used with the selected markers. It is recommended that the number of elements in this list be one less than the number of levels to be used - see **Marker Shading** in **Contour Shading Technique** above. If only one value is set, it will apply to all shading bands

Contour Shade Marker Table

Specifies the list of markers to be used. It is recommended that the number of elements in this list be one less than the number of levels to be used - see **Marker Shading** in **Contour Shading Technique** above. If only one value is set, it will apply to all shading bands

Contour Shade Method

Specifies the shading method to be used. Only available for **Contour Shade Technique** set to **Polygon Shading**. Options include :

- **Dot** - polygons are filled with dots of varying density, size and colour
- **Hatch** - polygons are filled with varying coloured hatched shapes
- **Area Fill** - solid shading which will completely fill the polygons

Contour Shade Dot Method

Specifies the method to apply when drawing dots. Options are **Old** (MAGICS "traditional" way) and **New** (uses texture mapping with OpenGL). Requires **Contour Shade Method** set to **Dot**.

Contour Shade Dot Size

Specify the size of the dots in cm. Larger dot sizes use more CPU time. Requires **Contour Shade Method** set to **Dot**.

Contour Shade Angle

Specify the angle of the shading lines as degrees anticlockwise from the horizontal. Shading lines are parallel lines along which the dots are drawn. Requires **Contour Shade Method** set to **Dot**.

Contour Shade Min Level Density

Specifies the dot density in dots/cm² assigned to the shading band whose lower limit is the level specified in **Contour Shade Min Level**. Requires **Contour Shade Method** set to **Dot**.

Contour Shade Max Level Density

Specifies the dot density in dots/cm² assigned to the shading band whose lower limit is the level specified in **Contour Shade Max Level**. Requires **Contour Shade Method** set to **Dot**.

Contour Shade Hatch Thickness

Specifies the thickness of the hatch pattern lines. Enter values from 1 to 10. Requires **Contour Shade Method** set to **Hatch**.

Contour Shade Hatch Density

Specifies the density of the hatch pattern lines (number of lines per cm). Requires **Contour Shade Method** set to **Hatch**.

Contour Shade Hatch Index

Specifies the type of hatch pattern to be used. Accepting the default (1) gets you a sequence of different patterns; any other value will apply the same pattern type (see Figure III-6) across the fields. Requires **Contour Shade Method** set to **Hatch**.

Figure III-6 : Hatch patterns available for contouring

Contour Shade Cell Method

Specifies the plotting method for shade cells. Options are **Interpolate** (derived from interpolation of neighbouring grid points) and **Nearest** (derived from value of nearest neighbour). Requires **Contour Shade Technique** set to **Cell Shading**.

Contour Shade Cell Resolution

Specifies the number of cells per cm. Requires **Contour Shade Technique** set to **Cell Shading**.

Contour Shade Label Blanking

Toggles blanking of the shaded contour labels **On** or **Off**. When **On**, no shading takes place in the area surrounding the contour labels and highs and lows.

Contour Line Plotting

Toggles between **Not Split** and **Split** options. The **Split** option allows you to plot values above/below a given value (which you specify in **Contour Split Level**) with different contour styles (defined by the **Contour Below...** and **Contour Above...** parameters). The level at which you split the contouring may also be plotted differently (via the **Contour Split...** parameters).

Contour Line Style

Specifies the line style to be used in the contouring. Only available if **Contour Line Plotting** is set to **Not Split**.

Contour Line Thickness

Specifies the line thickness to be used. Varies from 1 (thinner) to 10 (thicker). Only available if **Contour Line Plotting** is set to **Not Split**.

Contour Line Colour

Specifies the line colour to be used. Only available if **Contour Line Plotting** is set to **Not Split**.

Contour Highlight

Toggles the contour highlighting **On** or **Off**. Highlighting means plotting some contour lines in a different style from the others.

Contour Highlight Frequency

Specifies the frequency of highlighted contour lines - every n^{th} line is highlighted. Only available if **Contour Highlight** is set to **On**.

Contour Highlight Style

Specifies the style of the highlighted contour lines - you can choose from 5 styles. Only available if **Contour Highlight** is set to **On** and if **Contour Line Plotting** is set to **Not Split**.

Contour Highlight Thickness

Specifies the thickness of the highlighted contour lines. Varies from 1 (thinner) to 10 (thicker). Only available if **Contour Highlight** is set to **On** and if **Contour Line Plotting** is set to **Not Split**.

Contour Highlight Colour

Specifies the colour of the highlighted contour line. Only available if **Contour Highlight** is set to **On** and if **Contour Line Plotting** is set to **Not Split**.

Contour Split Level

Specifies the numerical value at which you want the contouring to be split between two styles. You can then apply a set of contour specifications to the data values above this value and another to the values below (see **Contour Below Options** and **Contour Above Options**). Only available if **Contour Line Plotting** is set to **Split**.

Contour Below Options

A set of six parameters that specify the plotting of the contour lines and contour highlight lines for the values below **Contour Split Level**. Includes **Contour Below Line Style**, **Thickness** and **Colour** and **Contour Below Highlight Style**, **Thicknes** [note the single 's' due to a limitation in the length of parameter names] and **Colour**). Only available if **Contour Line Plotting** is set to **Split**.

Contour Split Line Plot

Toggles the plotting of the contour line at the value specified in **Contour Split Level** (split contour line) **On** or **Off**. Only available if **Contour Line Plotting** is set to **Split**.

Contour Split Options

A set of three parameters that specify the plotting of the split contour line (**Contour Split Line Style**, **Thickness** and **Colour**). Only available if **Contour Line Plotting** is set to **Split**.

Contour Above Options

A set of six parameters that specify the plotting of the contour lines and contour highlight lines for the values above **Contour Split Level**. Includes **Contour Above Line Style**, **Thickness** and **Colour** and **Contour Above Highlight Style**, **Thicknes** [note the single 's' due to a limitation in the length of parameter names] and **Colour**. Only available if **Contour Line Plotting** is set to **Split**.

Contour Label

Toggles plotting of contour labels **On** or **Off**. When **Off**, the following seven **Contour Label *** parameters become unavailable.

Contour Label Frequency

Specifies the frequency of labelled contour lines - every n^{th} line is labelled

Contour Label Quality

Specifies quality of labels (**Low**, **Medium**, **High**)

Contour Label Height

Specifies the size of the contour labels (in cm)

Contour Label Colour

Specifies the colour of the labels

Contour Label Type

Specifies the type of label - you can specify **Number**, **Text** or their combination (**Both**). The former may be used if you want to plot both the contour value and the units

Contour Label Format

Specifies the number format of the contour label. Available only for **Contour Label Type** set to **Number** or **Both**. Use FORTRAN style format specifiers (e.g. **I4** for a 4 digit integer, **F5.1** for a $nnn.n$ number)

Contour Label Text

Specifies the text of the contour label. Available only for **Contour Label Type** set to **Text** or **Both**

Contour Hilo

Specifies plotting of the maxima and minima - Plot both (**On**), none (**Off**), maxima only (**Hi**) or minima only (**Lo**). Plotting of maxima and minima uses their values or the letters **H** and **L**. The following 14 parameters control this plotting and are only available if this option is set to **On**, **Hi** or **Lo**

Note that the extrema plotted are those of the fitted function, so they may not coincide with a grid point. You can thin the density of extrema plotting via the parameter **Contour Hilo Suppress Radius**

Contour Hilo Min Value

Suppress plotting of maxima and minima below this value

Contour Hilo Max Value

Suppress plotting of maxima and minima above this value

Contour Hilo Suppress Radius

Suppresses plotting of extrema whose positions are less than a certain distance away from the position of the plotted element. Further details can be found in the section *Contour Maxima/Minima* in the *MAGICS Users Guide* (<http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>)

Contour Hilo Reduction Radius

Thins out extrema whose positions are less than a certain distance away from the position of the plotted element. Further details can be found in the section *Contour Maxima/Minima* in the *MAGICS Users Guide* (<http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>)

Contour Lo Colour

Specifies colour of plotted minima. Available for **Contour Label** set to **On** or **Lo**.

Contour Hi Colour

Specifies colour of plotted maxima. Available for **Contour Label** set to **On** or **Hi**.

Contour Lo Min Value

Specifies a value below which plotting of minima is suppressed.

Contour Lo Max Value

Specifies a value above which plotting of minima is suppressed.

Contour Hi Min Value

Specifies a value below which plotting of maxima is suppressed.

Contour Hi Max Value

Specifies a value above which plotting of maxima is suppressed.

Contour Hilo Quality

Specifies print quality of labels. Options are **Low**, **Medium**, **High**

Contour Hilo Height

Specifies size of labels, in cm

Contour Hilo Type

Specifies the type of extrema plotting - **Text**, **Number** or **Both**

Contour Hilo Format

Specifies the extrema number format. Use FORTRAN style format specifiers (e.g. **I4** for a 4 digit integer, **F5.1** for a **nnn.n** number). Available only if **Contour Hilo Type** set to **Number** or **Both**.

Contour Lo Text

Specifies the text used to plot a minimum. Available for **Contour Hilo Type** set to **Text** or **Both**.

Contour Hi Text

Specifies the text used to plot a maximum. Available for **Contour Hilo Type** set to **Text** or **Both**.

Contour Hilo Marker

Toggles marking of the extrema with a symbol **On / Off**. Can be additional to any plotting defined in the **Contour Hilo** set of parameters above

Contour Hilo Marker Index

Specifies type of marker to be used. You have 5 symbols to choose from

Contour Hilo Marker Height

Specifies size of marker in cm.

Contour Hilo Position Write

Toggles the writing of the extrema to a geopoints file whose filename is specified by **Contour Hilo Position File Name**. Only valid if **Contour Hilo** is not **Off**. Depending on the value of **Contour Hilo**, the maxima (**Hi**), minima (**Lo**) or both (**On**) will be written to the file. Parameters which perform filtering on the extrema (such as **Contour Lo Min Value**) are not considered in the writing of the file. An error message is generated if the file cannot be written to, but the program will continue normally.

Contour Hilo Position File Name

Specifies the name of the geopoints file that will be written with the values of the extrema. Only valid if **Contour Hilo Position Write** is set to **On**. Note that it is advisable to provide a full path to the file if running Metview interactively.

Contour Grid Value Plot

Toggles plotting of the grid values **On / Off**. This option (if set) allows the plotting of the actual values of a field at the exact location of the values. Alternatively, markers may be plotted to indicate the positions of the grid points. The set of parameters which follow, specify the plotting details - format, size and colour of the values/markers. The range of values/markers plotted may be determined by the parameters **Input Field Suppress Above** and **Input Field Suppress Below**

Contour Grid Value Plot Type

Specifies whether to plot the **Value**, **Marker** or **Both** at each grid point.

Contour Grid Value Type

MAGICS expands GRIB fields into a regular matrix, even fields coded in quasi-regular grids. The regular grid values are plotted if this parameter is set to **Normal**, the quasi-regular grid values if this parameter is set to **Reduced**

Contour Grid Value Colour

Specifies the colour to be used for plotting the grid values

Contour Grid Value Format

Specifies the numerical format to be used for plotting the grid values. Use FORTRAN style format specifiers (e.g. I4, F5.1).

Contour Grid Value Height

Specifies the size of the numbers used in plotting the grid values in cm.

Contour Grid Value Quality

Specifies the print quality of the grid values.

Contour Grid Value Marker Colour

Specifies the colour to be used for plotting the grid markers.

Contour Grid Value Marker Index

Specifies the symbol used to mark each grid point.

Contour Grid Value Marker Height

Specifies the size of the symbols used in plotting the grid markers.

Contour Grid Value Marker Qual

Specifies the print quality of the grid markers.

Contour Grid Value Format

Specifies the numerical format to be used for plotting the grid values. Use FORTRAN style format specifiers (e.g. I4, F5.1).

Contour Grid Value Min

Specifies the minimum value that will be plotted at the grid points.

Contour Grid Value Max

Specifies the maximum value that will be plotted at the grid points.

Contour Grid Value Lat Frequency

Specifies the latitudinal frequency with which values are plotted at grid points. For example, a value of 3 causes values to be plotted at every third grid point, and the default value of 1 causes values to be plotted at every point in the latitude direction.

Contour Grid Value Lon Frequency

Specifies the longitudinal frequency with which values are plotted at grid points. For example, a value of 2 causes values to be plotted at every second grid point, and the default value of 1 causes values to be plotted at every point in the longitude direction.

Input Field Gradient Control

Parameter used in the control of the CONICON contouring method gradient estimation. CONICON requires the gradients (derivatives) along the longitude and latitude directions at each grid point. MAGICS calculates these internally. For some fields (with flat areas or not differentiable in both directions) the default method may not provide the best results. Note that the gradients only affect the estimation of the field within a grid cell. The grid point values remain unaffected and are always reflected in the contouring by CONICON.

The following choices are available :

- **Least** - Calculates the gradient as the minimum of the three gradients derived from the point and its two neighbours. Many extrema within grid cells will be weaker or removed
- **Flat Only** - Sets the gradient to 0 if the minimum of the three gradients derived from the point and its two neighbours is 0. Many extrema on the boundary of a flat area disappear. Useful for precipitation and relative humidity fields
- **Off** - Default

Input Field Gradient Limitation

This parameter sets limits to the value of the gradients derived for the CONICON contouring method.

- **Minimum** - gradient is set to 0, when values of the field are equal to or below the value specified in **Contour Min Level**

- **Maximum** - gradient is set to 0, when values of the field are equal to or above the value specified in **Contour Max Level**
- **Both** - Combines both previous options
- **Off** - Default

Input Field Suppress Above

Suppresses grid values above the specified value. Note that this is not the same as limiting the contour of values above a given value (this is the role of **Contour Max Level**). Gaps in the field resulting from this command will be contoured. Not available for **Contour Method** set to **Linear**

Input Field Suppress Below

Suppresses grid values below the specified value. Note that this is not the same as limiting the contour of values below a given value (this is the role of **Contour Min Level**). Gaps in the field resulting from this command will be contoured. Not available for **Contour Method** set to **Linear**

Grib Spectral Resolution

MAGICS expands GRIB fields into a regular grid. This parameter specifies the resolution of the regular grid. This has no effect if the field is already in a regular grid (LL or GG). See the MARS Retrieval parameter **Grid** in page III-26.

Grib Scaling of Retrieved Fields

Toggles the scaling of the retrieved fields **On/Off**. Fields which are retrieved from MARS or derived from other fields are in SI units. MAGICS performs a unit conversion (scaling) on the retrieved fields that it plots, converting from these SI units to units of customary meteorological usage - e.g. Pressure from Pa to hPa/mb, Temperature from K to °C.

Grib Scaling of Derived Fields

Toggles the scaling of the derived fields **On/Off**. Any field you derive is in SI units, so set this to **On** to convert to meteorological style units. E.g. :

- If you retrieve two temperature fields, they are plotted in °C. If you derive a mean temperature from them, it will be plotted in K if you do not scale the derived field.
- Precipitation fields are cumulative fields plotted in mm - if you subtract two consecutive ones to obtain the precipitation for the time step between them, you will plot a field in m if you do not scale the derived field.

Grib Missing Value Indicator

Assigns a value to missing values in fields. A judicious choice of this value allows you to exclude missing values from the plotting, using parameters **Contour Max Value**, **Contour Min Value** and **Input Field Suppress Above** and **Input Field Suppress Below**.

Grib Text

MAGICS can generate text automatically from the elements in the GRIB header. This parameter toggles such text generation **On/Off**

Grib Text Split

Allows the title text generated by MAGICS to be split into 2 lines.

Grib Text Units

Adds the units of the contours to the title generated by MAGICS. This parameter toggles this feature **On/Off**

Grib Text Experiment

Adds the experiment version of the field to the title generated by `MAGICS`. This parameter toggles this feature **On/Off**

Grib Text Plot Type

Adds the plot type (eg image, wind, contour) to the title generated by `MAGICS`. This parameter toggles this feature **On/Off**

Dynamic Memory Allocation Factor

Occasionally, complex contour plots will require more memory than has been allocated. In this case, Metview will issue a warning. This parameter allows you to adjust the memory allocation so that complex contour plots will succeed. The default value is 100, which causes no change in the dynamic memory allocation. If your plot fails due to memory allocation errors, try increasing this value by 20 and try again. If the plot still fails, increase the parameter by another 20 and so on until it succeeds. In order to avoid unnecessary system slowdown, it is best to set this value to the lowest at which it will work, hence the suggested incremental approach.

Actions on the Contour Icon

The actions available for the Contour icon are *Edit*, *Duplicate* and *Delete*.

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

WIND PLOT



This is the visual definition used for plotting winds. Note that only gridded data can be plotted with this visual definition - vectorial geopoints data use the Symbol visual definition (see "Symbol" on page III- 217). It is the translation of the MAGICCS wind plotting directives into a Metview icon. The macro language equivalent is `pwind()`.

Further details, not presented in this manual can be found in Chapter 6 ("Wind Field Plotting") of the MAGICCS Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>.

The Wind Plot Editor

Most of the parameters of the Wind Plot editor belong to one of three groups - Wind Flags, Wind Arrows, Wind Streamlines.

Depending on which type of plotting element you choose for your wind data, one of these groups will be available as a block. If you wish to use isotachs (contour lines of wind speed) to plot your wind data, you must use the Contour icon (see "Contour Data Transformation" on page III- 184).

The parameters are assigned sensible defaults, to enable you to produce good quality plots with minimum changes to the default values.

Legend

Toggles the user defined legend **On** or **Off**.

Legend User Text

Type the text you want to go with the legend. Unavailable if **Legend** set to **Off**.

Legend Entry

Drop a Legend Entry icon (see "Legend Entry" on page III- 231). This has no effect unless **Legend** is set to **On**.

Wind Thinning Factor

Specifies the thinning factor. Depending on the length of the arrows or flags, the grid size of the data and geographical projection, arrows and wind flags may overlap. Thinning reduces the density of the data to minimise/remove overlapping. For latitudes beyond 60 degrees thinning of the wind field is automatic in polar stereographic projection

For data in cylindrical projection, the specified thinning factor is rounded up to the next whole number n and wind plotting elements (arrows or flags) are plotted only every n regular grid points.

For data in polar stereo the thinning factor is the specified number multiplied by the projected longitude step - points closer than this distance are ignored

Wind Field Type

Specifies the type of plotting element - **Arrows**, **Flags** or **Streamlines**. This option decides which of the group of parameters following, becomes active

- **Arrows** - Wind is represented by an arrow whose length is proportional to the wind speed, pointing in the direction of the wind at the grid position
- **Flags** - Wind is represented by a WMO standard wind flag symbol, pointing in the direction from which the wind is coming (barbs and pennants indicate the wind speed)
- **Streamlines** - Wind is represented by a line everywhere tangent to the wind vector

Wind Flag Length

Specifies the length of the wind flag in cm

Wind Flag Mode

Options are **Normal**, **Off Level**, **Off Time**. When you plot a wind field on top of a pre-existing one, it is desirable to distinguish them. This is done via this parameter which controls the appearance of the wind flag :

- **Off Level** - if the existing wind field is at a different level, this setting draws the entire wind flag shaft as a dashed line.
- **Off Time** - if the existing field is at a different time, this setting draws the wind flag shaft as a dashed line up to the barbs and pennants.
- **Normal** - Draws the flag shaft as a solid line

Wind Flag Origin Marker

Specifies the form of the flag origin point. Options are **Circle**, **Dot**, **Off**.

- **Circle** - plots origin as a circle; wind flag shaft starts at perimeter of circle
- **Dot** - plot origin as a dot; wind flag shaft starts a short distance from the dot
- **Off** - origin not plotted; wind flag shaft starts at the origin point. Reverts to **Dot** option if **Wind Flag Calm Indicator** is set to **On**.

Wind Flag Min Speed

Specifies the value below which plotting of wind flags is suppressed

Wind Flag Max Speed

Specifies the value above which plotting of wind flags is suppressed

Wind Flag Calm Indicator

Toggles the wind calm indicator **On** or **Off**.

Wind Flag Calm Indicator Size

Specifies the size of the wind calm indicator in cm

Wind Flag Thickness

Specifies the thickness of the wind flag shaft (not the barbs and pennants) from **1** to **10** (thickest)

Wind Flag Colour

Specifies the colour of the wind flag

Wind Flag Cross Boundary

If **On**, wind flags are truncated if they cross the display window/subpage border

Wind Arrow Unit Velocity

Specifies the wind speed represented by a unit vector. The higher this value the shorter the wind arrows. If you set this to 0.0, MAGICS plots wind arrows by setting the unit vector to the maximum wind speed in the field

Wind Arrow Min Speed

Specifies the value below which plotting of wind arrows is suppressed

Wind Arrow Max Speed

Specifies the value above which plotting of wind arrows is suppressed

Wind Arrow Calm Indicator

Toggles the wind calm indicator **On** or **Off**.

Wind Arrow Calm Indicator Size

Specifies the size of the wind calm indicator in cm

Wind Arrow Calm Below

Specifies wind speed below which wind is considered calm

Wind Arrow Thickness

Specifies the thickness of the wind arrow shaft from **1** to **10** (thickest). Dimensions of the arrow head are adjusted accordingly

Wind Arrow Colour

Specifies the colour of the wind arrow shaft

Wind Arrow Cross Boundary

If **On**, wind arrows are truncated if they cross the plotwindow/subpage border

Wind Arrow Head Index

Specifies the style and shape of the wind arrow head

Wind Arrow Origin Position

Specifies the positioning of the wind arrow relative to the grid point. Options are **Tail** and **Centre**

- **Tail** - the tail of the wind arrow is plotted at the wind origin
- **Centre** - the centre of the wind arrow is plotted at the wind origin

Wind Arrow Legend

Toggles the wind arrow legend box **On/Off**. A wind arrow legend box with the unit vector and its unit value is plotted just outside the subpage/plotwindow. If **Off**, the following wind arrow related parameters become unavailable

Wind Arrow Legend X Position

Specifies the X coordinate of the lower left hand corner of the legend box

Wind Arrow Legend Y Position

Specifies the Y coordinate of the lower left hand corner of the legend box

Wind Arrow Legend X Length

Specifies the X length of the wind legend box

Wind Arrow Legend Y Length

Specifies the Y length of the wind legend box

Wind Arrow Legend Blanking

Toggles blanking of the wind legend box **On/Off**. If **On**, everything behind the legend box is blanked - this only applies when the legend box overlaps with the display window/subpage

Wind Streamline Min Density

Specifies the number of streamlines per sq cm of a plotted page. This controls the density of streamlines in the plot

Wind Streamline Min Speed

Specifies the minimum wind speed below which streamline plotting is stopped. The value must be lower than 6m/s

Wind Streamline Closeness Check

Toggles streamline closeness check **On/Off**

Wind Streamline Closeness Margin

Specifies in cm how close a streamline can get to another before being truncated. If **Wind Streamline Closeness Check** is **Off**, they are allowed to coalesce

Wind Streamline Thickness

Specifies the thickness of the wind streamline from **1** to **10** (thickest).

Wind Streamline Colour

Specifies the colour of the wind streamline

Wind Streamline Style

Specifies the style of the wind streamline

Wind Streamline Calm Marker

Toggles the plotting of calm points On/Off. Calm points correspond to the wind speed minima in the wind field. Markers are different for anticyclonic and cyclonic minima. Asymptotic points are indicated by two short thin pieces of streamline, rather than a marker

Wind Streamline Marker Index

Specifies the type of marker to be used for the calm points. **1** is unfilled circle, **2** unfilled triangle, **3** is cross (default), **4** is an X and **5** unfilled diamond

Wind Streamline Marker Height

Specifies the height in cm of marker to be used for the calm points

Wind Streamline Arrow Head Index

Specifies the style and shape of the streamline arrow heads

Wind Streamline Arrow Head Length

Specifies the length of the streamline arrow heads

Input Field Suppress Above

Suppresses grid values above the specified value. In this case the effects are the same as limiting the plotting for values above a given value (by setting the **Max Speed** parameters)

Input Field Suppress Below

Suppresses grid values below the specified value. In this case the effects are the same as limiting the plotting for values below a given value (by setting the **Min Speed** parameters)

Grib Spectral Resolution

MAGICS expands GRIB fields into a regular grid. This parameter specifies the resolution of the regular grid. This has no effect if the field is already in a regular grid (LL or GG). See the MARS Retrieval parameter **Grid**

Grib Missing Value Indicator

Assigns a value to missing values in fields. A judicious choice of this value allows you to exclude missing values from the plotting, using parameters **Max/Min Speed** and **Input Field Suppress Above** and **Input Field Suppress Below**.

Grib Text Split

Allows the title text generated by MAGICS to be split into 2 lines.

Grib Text Units

Adds the units of the wind arrows/flags/streamlines to the title generated by MAGICS. This parameter toggles this feature **On/Off**

Grib Text Experiment

Adds the experiment version of the field to the title generated by MAGICS. This parameter toggles this feature **On/Off**

Grib Text Plot Type

Adds the plot type (eg image, wind, contour) to the title generated by MAGICS. This parameter toggles this feature **On/Off**

Actions on the Wind Plot Icon

The actions available for the Wind Plot icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

OBSERVATION PLOT



This is the visual definition used for plotting observations. It is the translation of the `MAGICS` observation plotting directives into a Metview icon. The macro language equivalent is `pobs()`.

Further details, not presented in this manual can be found in Chapter 7 ("Observation Plotting") of the `MAGICS` Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>.

Main Observation Plot Features

The Observation Plot editor comprises three parts. The first part contains general specification about the plotting (from **Obs Type List** to **Obs Position Only**). The second part (from **Obs Station Ring** to **Obs Station Height Colour**) specifies the plotting of Observation data. The third part specifies the plotting of Observation Feedback data (**Fb**). Note that the second and third parts are exclusive, i.e. you either use one or the other, depending on which set you want to plot.

Analysis BUFR Feedback files describe the history of observations as they pass through the ECMWF data assimilation system and they exist for two different analysis schemes - 3DVAR (3D Variational Analysis) and 4DVAR (4D Variational Analysis). Currently, the variables used in the feedback are wind, temperature, height, relative humidity, specific humidity and surface pressure. Each observation in the feedback files consists of some or all of the following :

- Original BUFR message in full
- Substitution values, if any actually given, otherwise missing
- Blacklisted data, if any data has been blacklisted, otherwise missing.
- Analysis variables presented to the analysis
- Variational analysis flags :
 - FG, first guess flags
 - DP, departure flags
 - 3D and 4D, quality control flags
 - BL, blacklist flags

The parameters are assigned sensible defaults, to enable you to produce good quality plots with minimum changes to the default values.

The Observation Plot Editor

As mentioned in the previous section, the input parameters to this icon have been divided into three groups for easier navigation :

- Parameters for general specification of observation plotting

- Parameters for plotting of individual observation elements
- Parameters for plotting of observation feedback

General specification of Observation plotting

Here are listed the parameters governing the basic aspects of observation plotting. You can filter out observations according to type, time, confidence level and set basic plotting characteristics - e.g. size, quality and density of plotted items.

Obs Type List

Specifies the Observation Types, one of SYNOP, AIREP, SATOB, DRIBU, TEMP, SOILT, PILOT, SATEM. Enter chosen type in the text field or select it from the selection help list. Only observations of the selected types are plotted. If no types are selected (the default setting), all available types will be plotted.

Obs Code Type List

Specifies the code type for the selected observation type. Within each observation type, there are various code types. Enter chosen code type in the text field or select it from the selection help list. Only observations of the selected code types are plotted. If no code types are selected (the default setting), all available code types will be plotted.

Obs Identifier List

Specifies a list of station identifiers, e.g. land station identifiers and ship identifiers. Do not use to pass satellite identifiers (see next)

Obs Satellite Number List

Specify the satellite identifier number(s) for both SATOB and SATEM observations. Enter chosen numbers in the text field or select satellite from the selection help list.

Obs Confidence Check

Toggles observation confidence checking **On/Off**. If this is set to **On**, then only observations whose confidence level is lower than the value specified below have their station ring plotted.

Obs Confidence Parameter

Specifies the parameter to undergo confidence check. You can select a particular one out of 6 or all by selecting **Any**.

Obs Confidence Level

Specifies the confidence level to attach to the confidence check (1 - 100).

Obs Date Check

Toggles the date check **On/Off**. Only observations for the date specified below will be plotted

Obs Date

Specify the date (according to a chosen date format) at which observations will be plotted

Obs Date Tolerance

Specifies a date range for plotting observations. All the observations within **Date +/- Date Tolerance** will be plotted

Obs Time Check

Toggles the time check **On/Off**. Only observations for the time specified below will be plotted

Obs Time

Specifies the time (HHMM) for which observations will be plotted

Obs Time Tolerance

Specifies a time range (HHMM) for plotting observations. All the observations within **Time +/- Time Tolerance** will be plotted

Obs Identification

Toggles plotting of the observation identification **On/Off**

Obs Identification Colour

Specifies the colour to use in plotting the observation identification

Obs Identification Type

Specifies which parameter to be used in the identification of the observation, the **Station Identifier**, the **Code Type** or the **Observation Type**.

Obs Distance Apart

Specifies the minimum distance apart between plotted observations, in cm

Obs Size

Specifies the size of the plotted variables in cm

Obs Quality

Specifies the quality of the observation plotting

Obs Rotation

Toggles observation rotation **On/Off**. Only applicable for observations plotted in polar stereographic projection. If **On**, observations are plotted everywhere tangential to the latitude lines

Obs Position Only

If **On**, plots only the position of the observations

Plotting elements for Observation

The remaining parameters from **Obs Station Ring** to **Obs Station Height Colour**, specify the plotting of Observation data. They are self explanatory and arranged as a sequence of groups of observation element and plotting elements (colour mostly). To operate with these, just toggle the plotting of the required observation **On** or **Off** and then select the colour or any other plotting element with which you want it plotted.

Plotting elements for Feedback (Fb)

Obs Fb

Specifies whether to plot Observation Feedback (**Off**) and the type of Feedback (**3dvar**, **4dvar**). Any choice other than **Off** will make all previous standard observation plotting parameters become unavailable.

Obs Fb Legend

Toggles feedback plot legend abilities **On/Off**. This legend will plot extra information when plotting rejected data

Obs Fb Type

Specifies the plotting option of feedback data. Options are :

- **Reject Report** - reports rejected by FG, DP, 3D or 4D
- **Reject Variables** - variables rejected by FG, DP, 3D or 4D
- **Flagged But Used** - variables flagged by FG, DP, 3D or 4D but used in the analysis
- **Used** - variables used in the analysis
- **Not Used** - variables not used in the analysis
- **Blacklist Report** - reports containing blacklisted data
- **Blacklist Variables** - blacklisted variables
- **All** - all variables, including those rejected and/or blacklisted

Obs Fb Station Ring Height

Specifies size of the Fb station ring to be plotted (cm)

Obs Fb Min Error Flag

Specifies the lowest error flag value for which data is to be plotted (see also **Obs Fb Substitutes** below)

Obs Fb Error Flag0 Colour

Specifies the colour assigned to observations at error severity 0 (correct)

Obs Fb Error Flag1 Colour

Specifies the colour assigned to observations at error severity 1 (possibly incorrect)

Obs Fb Error Flag2 Colour

Specifies the colour assigned to observations at error severity 2 (probably incorrect)

Obs Fb Error Flag3 Colour

Specifies the colour assigned to observations at error severity 3 (Incorrect)

Obs Fb Source

Toggles plotting of the source of rejection **On/Off**

Obs Fb Source Type

Specifies a specific source of rejection of the reports (RDB, ML, FG, AN, DP, BL, 3D, 4D, ALL)

Obs Fb Level

Specifies the required pressure level. See **Obs Level Tolerance** below.

Obs Fb Level 2

Specifies a second pressure level (thickness only)

Obs Fb Level Tolerance Above

Specifies an upper tolerance for the pressure level. This parameter, together with the following one and **Obs Fb Level** allow you to select a pressure range to be searched for rejected data - the level containing the requested data nearest to the value specified in **Obs Fb Level** is selected.

Obs Fb Level Tolerance Below

Specifies a lower tolerance for the pressure level. See above

Obs Fb Departures

Toggles plotting of departure values instead of analysis values **On/Off**

Obs Fb Departures Type

Specifies where the departures are measured from. For **Obs Fb** set to **Oi**, the options are :

- **FG** - First Guess
- **AN** - Analysis
- **IA** - Initialised Analysis

The other options apply for **Obs Fb** set to **3dvar** or **4dvar** :

- **OBS-FG** - departures from the high resolution model, T213
- **OBS-FG2** - initial departures from the low resolution model, T63
- **OBS-FG3** - final departures from the low resolution run, T63
- **OBS-AN** - final departures from the final analysis

Obs Fb Substitutes

Toggles **On/Off** plotting of substitute values. The parameter **Obs Fb Min Error Flag** (see above) should be set to restrict the plotting to data where the error rejection flag is greater or equal to the set value

Obs Fb Height

Toggles plotting of the parameter **On/Off**

Obs Fb Pressure

Toggles plotting of the parameter **On/Off**

Obs Fb Relative Humidity

Toggles plotting of the parameter **On/Off**

Obs Fb Specific Humidity

Toggles plotting of the parameter **On/Off**

Obs Fb Temperature

Toggles plotting of the parameter **On/Off**

Obs Fb Wind

Toggles plotting of the parameter **On/Off**

Obs Fb Wind Flag Length

Specifies the length of the wind flag in cm

Obs Fb Radiances

Toggles plotting of TOVS radiances **On/Off**.

Obs Fb Radiances Type

Specifies the type of radiances to plot. Options are **1dvar**, **3dvar**, **4dvar**

Obs Fb Rad Channel Number

Specifies the number of the TOVS channel for which radiances are to be plotted

Obs Fb Radiances Channel Plot

Toggles plotting of the TOVS channel number **On/Off**

Obs Fb Radiances Departures

Toggles plotting of TOVS radiance departures **On/Off**. Note that values plotted for radiances departures are the actual values multiplied by 10

Obs Fb Rad Departures Type

Specifies where the departures are measured from. For **Obs Fb Radiances Type** set to **1dvar**, the options are **OBS-FG, ALL, BIAS, STDEV**. For **Obs Fb Radiance Type** set to **3dvar** or **4dvar**, valid values are **OBS-FG, OBS-FG2, OBS-FG3, OBS-AN, ALL**.

Obs Fb Scatterometer Winds

Toggles plotting of scatterometer winds **On/Off**. Two wind values are provided arising from almost opposite directions

Obs Fb Scatterometer Check

Toggles checking of scatterometer winds closest to analysed winds **On/Off**. If this check is **Off**, both winds are plotted, if **On** only the wind closest to the analysis is plotted

Obs Fb Ssmi Water Precipitable

Toggles plotting of SSMI precipitable water **On/Off**

Obs Fb Ssmi Liquid Water Path

Toggles plotting of SSMI liquid water path **On/Off**

Obs Fb Ssmi Type

Specifies the type of SSMI Fb Observation data. Options are **1dvar** and **Regression**

Obs Fb Ssmi Wind Strength

Toggles plotting of SSMI wind strength **On/Off**

Obs Fb Ssmi Failure Flag

Toggles plotting of SSMI failure flag **On/Off**

Obs Bufr Section2 Use

Toggles **On/Off** the usage of Bufr Section 2 information. ECMWF uses this section to store observation handling information, thereby avoiding decoding of BUFR messages, so this **On** by default. You can still set this to **Off** when plotting ECMWF data but then decoding will take place and plotting will be slower. For data originating outside ECMWF this parameter will have to be set to **Off**, since other institutions use BUFR Section 2 in different ways.

Obs Text

Actions on the Observation Plot Icon

The actions available for the Observation Plot icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

COASTLINES



This is the visual definition used for plotting of geographical details (e.g. grid, coastline). It is the translation of the MAGICS coastline plotting directives into a Metview icon. The macro language equivalent is `pcoast()`.

The coastline data used by Metview is derived from the GSHHS - see Wessel, P., and W. H. F. Smith, A Global Self-consistent, Hierarchical, High-resolution Shoreline Database, J. Geophys. Res., 101, #B4, pp. 8741-8743, 1996.

Further details, not presented in this manual can be found in Chapter 3 ("Layout, Mapping and Coastlines") of the MAGICS Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>.

The Coastlines Editor

The editor consists of 5 groups of parameters specifying details for the plotting of the subpage frame, map coastline, map grid, map label and land-sea colours. The parameters are assigned sensible defaults, to enable you to produce good quality plots with minimum changes to the default values.

Map Coastline

Toggles the plotting of the coastline **On/Off**. If **On**, the following Map Coastline parameters become active :

Map Coastline Parameters

Including **Map Coastline Resolution**, **Map Coastline Style**, **Map Coastline Colour** and **Map Coastline Thickness**, parameters specifying the plotting characteristics of the coastline

Map Grid

Toggles the plotting of the grid lines **On/Off**. If **On**, the following Map Grid parameters become active :

Map Grid Parameters

Including **Map Grid Line Style**, **Map Grid Colour** and **Map Grid Thickness**, parameters specifying the plotting characteristics of the grid lines. A further set of grid line related parameters follow - these control the position and spacing of the grid lines in both dimensions :

Map Grid Latitude Reference

Specifies the starting (top) latitude for the plotting of the grid lines

Map Grid Longitude Reference

Specifies the starting (left) longitude for the plotting of the grid lines

Map Grid Latitude Increment

Specifies the latitude increment for the plotting of the grid lines

Map Grid Longitude Increment

Specifies the longitude increment for the plotting of the grid lines

Map Label

Toggles the plotting of the map lat/long labels **On/Off**.

Map Label Latitude Frequency

Specifies the plotting frequency of map latitude labels. Enter a number *n* - labels will be plotted every *n* latitude lines

Map Label Longitude Frequency

Specifies the plotting frequency of map longitude labels. Enter a number *n* - labels will be plotted every *n* longitude lines

Map Label Height

Specifies the height of the map labels in cm

Map Label Quality

Specifies the plotting quality of the map labels

Map Label Colour

Specifies the colour of the map labels

Map Coastline Land Shade

Toggles the colouring of the land areas **On/Off**

Map Coastline Land Shade Colour

Specifies the colour for the plotting of the land areas

Map Coastline Sea Shade

Toggles the colouring of the sea areas **On/Off**

Map Coastline Sea Shade Colour

Specifies the colour for the plotting of the sea areas

Map Coastline Path

Allows an alternative coastline file to be used. The value of this parameter should be a path with no filename; the filename is calculated as: NOAA_x_r8.bin (64-bit FORTRAN Reals) or NOAA_x.bin (32-bit FORTRAN Reals) where 'x' is L, M or H - the parameter **Map Coastline Resolution** determines which of these is selected. The coastline file should be generated using the same flags as the MAGICS library in Metview so that the Real values are of the correct size.

Actions on the Coastlines Icon

The actions available for the Coastlines icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

TEXT PLOT



This is the visual definition used for the plotting of title text in plots. Title text is always placed at the top of the plot. This icon also contains some parameters for adjusting the appearance of the legend. Freely positioned text is specified by means of the Annotation icon (see "Annotation" on page III- 215). Text implements some MAGICS text plotting directives as a Metview icon. The macro language equivalent is `ptext()`.

Further details, not presented in this manual can be found in Chapters 14 ("Text Plotting") and 15 ("Legend Plotting") of the MAGICS Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>. See also "Extension of Automatic Title Text in MAGICS" for further details on the automatic generation of text.

The Text Plot Editor

The editor comprises parameters which specify the where and how of plotting the text and a long sequence of parameters which specify what is to be plotted in which line of text. Text is plotted as a block of *n* lines (up to a maximum of 10) in a text box, which may be user-defined or automatic.

Text Automatic

Specifies whether to use the automatic text generation from the metadata (**Yes/No**).

Text User

Specifies whether to use user-specified text (**Yes/No**).

Text Line N

A set of 10 parameters (*N* goes from 1 to 10). Only available for **Text User** set to **Yes**. Specify the text to be *stored* in text line number *N*. The maximum number of characters you can specify is 200, *including special characters not appearing in the printed page*. The text lines are stored and the parameters **Text First Line** and **Text Line Count** determine which lines are actually plotted

Text First Line

Specifies the number of the first line of text to be plotted. See also **Text Line N**

Text Line Count

Specifies the number of lines of user-specified text to be plotted in the text block. The maximum number is 10. Enter the text for each line in the **Text Line N** parameters that follow.

Text Parameter Escape Character

The **Text Parameter Escape Character** allows the user to include parameter values within a text string. See the MAGICS Users Guide for more details.

Text Instruction Shift Character

Instruction Strings enable the user, amongst other things, to change the colour of text inside a text block and to plot complicated text strings such as mathematical equations. Instruction Strings are enclosed between two instances of the **Text Instruction Shift Character**, specified in this parameter. See the MAGICS Users Guide for more details.

Text Meta Field Escape Character

The **Text Meta Field Escape Character** allows descriptive fields to be automatically inserted into the text. For example, the string '!UNITS!' is replaced by a string describing the units applicable to the data being plotted.

Text Reduction Level

Specifies the level of detail to be presented in the title text from full details (0) to least details (2).

Text Mode

Toggles between the two modes of plotting text (**Title/Positional**). You can plot text as :

- **Title** - your text is automatically positioned above the subpage. The height and width of the box are automatically calculated
- **Positional** - you can place your text anywhere within the page via the parameters controlling the positioning and dimensions of the text box (see immediately below)

Text Box Parameters

Including **Text Box X Position**, **Text Box Y Position**, **Text Box X Length**, **Text Box Y Length**, these parameters control the positioning and dimensions of the text box. They are only active if **Text Mode** is set to **Positional**. The X and Y position of the text box is specified in cm from the page lower left corner

Text Box Blanking

Toggles blanking of the text box **On/Off**. If **On**, it will blank out the area defined by the text box before plotting the text

Text Reference Character Height

Specifies the height of the text lines in cm. However, the text size may be adjusted so as to fit the all the specified text lines into the current text block. Note that this is not the same as the height of the text characters. The height of the text characters in line N (in cm) is determined by the product of this parameter and the parameter Text Line Height Ratio N (see below)

Text Quality

Specifies the plotting quality of the text

Text Justification

Specifies the justification of the text

Text Colour

Specifies the colour of the text

Text Border

Toggles the plotting of the text border **On/Off**. If **On**, the following 3 parameters become active

Text Border Parameters

Including **Text Border Colour**, **Text Border Line Style** and **Text Border Thickness**, parameters specifying the plotting characteristics of the text border

Text Titles Table Use

Toggles use of the text table **On/Off**. If **On**, a table will be used in the generation of automatic text. If **Off**, the 'old' method will be adopted, using hard-coded text.

Text Legend Magics Style

Toggles between the two styles of Legend plotting. When **On**, the legend box will be positioned, by default, between the top of the subpage and the bottom of the title text block, if one exists. This is the 'old' style of legend plotting. When **Off**, a different set of parameters become active.

Text Legend Box Blanking

Toggles blanking of the legend box **On/Off**. If **On**, it will blank out the area defined by the legend box before plotting the text

Legend Positioning

Determines whether the legend should be positioned **Inside** or **Outside** the subpage. If **Automatic** is selected, it provides the default setting (**Outside**).

Legend Subpage Position

If **Legend Positioning** is **On**, the legend may be positioned at any of the four corners of the plot.

Legend Entry Plot Direction

Changes the order in which legend entries are displayed. If set to **Row**, the legend entries increase sequentially along the rows and then down the columns. If set to **Column**, the legend entries increase sequentially down the columns and then along the rows.

Legend Column Count

Specifies the number of columns displayed in the Legend box. If set to zero, the number of columns will be automatically determined according to the dimensions of the legend box and the available space.

Legend Box Mode

Specifies whether the legend box should be positioned automatically or manually. If set to **Automatic**, the legend will be placed at the right-hand edge of the subpage if there is room; otherwise it will be placed at the top of the subpage.

Legend Display Type

Specifies whether the legend should be **Disjoint** or **Continuous**.

Legend Box Parameters

Including **Legend Box X Position**, **Legend Box YPosition**, **Legend Box X Length** and **Legend Box Y Length**.

Legend Title

Specifies whether or not a title should be displayed for the legend.

Legend Title Text

Specifies the text to be used for the legend. Only available if **Legend Title Text** is set to **On**.

Legend Text Quality

Sets the legend text quality to **Low**, **Medium** or **High**.

Legend Text Maximum Height

Specifies the maximum height, in cm, of the legend text entries.

Legend Text Colour

Specifies the colour of the legend text entries.

Legend Box Blanking

Legend box blanking will blank out all the graphical output plotted previously within the legend box.

Legend Border

Toggles the legend border **On/Off**.

Legend Border Colour

Specifies the colour of the legend border.

Legend Border Line Style

Specifies the line style of the legend border.

Legend Border Thickness

Specifies the thickness of the legend border.

Legend Entry Maximum Height

The maximum height of a legend entry box in cm.

Legend Entry Minimum Height

The minimum height of a legend entry box in cm.

Legend Entry Maximum Width

The maximum width of a legend entry box in cm.

Legend Entry Minimum Width

The minimum width of a legend entry box in cm.

Legend Text Format

Specifies the automatic text format.

Legend Symbol Height Mode

Specifies whether the symbols in the legend box are displayed at the same height as the symbols on the plot (**As Map**) or as the text in the legend box (**As Text**).

Actions on the Text Plot Icon

The actions available for the Text Plot icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

ANNOTATION



This is the visual definition used for the plotting of freely positioned text in plots. Plot title text is specified by means of the Text icon (see "Text Plot" on page III- 211). Annotation implements some MAGICS text plotting directives as a Metview icon. The macro language equivalent is `annotation()`.

Further details, not presented in this manual can be found in Chapter 14 ("Text Plotting") of the MAGICS Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>. See also "Extension of Automatic Title Text in MAGICS" for further details on the automatic generation of text.

The Annotation Editor

The editor comprises parameters which specify the text to annotate a given user plot and its characteristics and location in the plot. Annotations are plotted as a block of *n* lines (up to a maximum of 10) in user defined boxes.

Text Line N

A set of 10 parameters (*N* goes from 1 to 10). Specify the text to be *stored* in text line number *N*. The maximum number of characters you can specify is 200, *including special characters not appearing in the printed page*. The text lines are stored and the parameters **Text First Line** and **Text Line Count** determine which lines are actually plotted

Text First Line

Specifies the number of the first line of text to be plotted.

Text Line Count

Specifies the number of lines of user-specified text to be plotted in the text block. The maximum number is 10. Enter the text for each line in the **Text Line N** parameters that follow.

Text Parameter Escape Character

The **Text Parameter Escape Character** allows the user to include parameter values within a text string. See the MAGICS Users Guide for more details.

Text Instruction Shift Character

Instruction Strings enable the user, amongst other things, to change the colour of text inside a text block and to plot complicated text strings such as mathematical equations. Instruction Strings are enclosed between two instances of the **Text Instruction Shift Character**, specified in this parameter. See the MAGICS Users Guide for more details.

Text Order

Specifies the order in which the lines of text are actually plotted

Text Box Parameters

Including **Text Box X Position**, **Text Box Y Position**, **Text Box X Length**, **Text Box Y Length**, these parameters control the positioning and dimensions of the text box. They are only active if **Text**

Mode is set to **Positional**. The X and Y position of the text box is specified in cm from the page lower left corner

Text Box Blanking

Toggles blanking of the text box **On/Off**. If **On**, it will blank out the area defined by the text box before plotting the text

Text Reference Character Height

Specifies the height of the text lines in cm. However, the text size may be adjusted so as to fit the all the specified text lines into the available space.

Text Quality

Specifies the plotting quality of the text

Text Justification

Specifies the justification of the text

Text Colour

Specifies the colour of the text

Text Border

Toggles the plotting of the text border **On/Off**. If **On**, the following 3 parameters become active

Text Border Parameters

Including **Text Border Colour**, **Text Border Line Style** and **Text Border Thickness**, parameters specifying the plotting characteristics of the text border

Actions on the Annotation Icon

The actions available for the Annotation icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

SYMBOL



This is the visual definition used for the plotting of symbols (number, text, marker, wind) at selected locations. It is the translation of the MAGICS symbol plotting directives into a Metview icon. The macro language equivalent is `psymb()`.

Further details, not presented in this manual can be found in Chapter 12 ("Symbol Plotting") of the MAGICS Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>.

The Symbol Editor

Symbol Type

Specifies the type of symbol to be used. Options are **Number**, **Text**, **Marker** and **Wind**. Each option disables some of the other parameters, e.g. **Wind** which leaves only **Symbol Colour** and the last two parameters available

Symbol Table Mode

Toggles the table mode **On/Off**. Symbol plotting may use one of two modes :

- **Table Mode** - in this mode you specify ranges of values and all the values within a given range are plotted with a symbol or text whose type, colour and height you specify
- **Individual Mode** - in this mode each value is plotted individually : either the value itself is plotted or a symbol or text is used. The symbol and text will have the same colour and height but different text strings or types of symbols may be used, provided their number matches the number of values to be plotted

When **Symbol Table Mode** is **On** the following parameter **Legend** is activated.

Legend

Toggles plotting of legend **On/Off**. This is only available for **Symbol Table Mode** set to **On**.

Legend Entry

Drop a Legend Entry icon (see "Legend Entry" on page III- 231). This has no effect and is unavailable unless **Legend** is set to **On**.

Symbol Position Mode

Specifies the positioning mode of the symbols. Options are :

- **Geographic** - applies to data in geographic coordinates
- **Paper** - applies to data whose position is specified in cm relative to the lower left corner of the subpage
- **Graph** - applies to data whose position is specified in x-axis, y-axis units

Symbol Border Check

Toggles border checking when plotting symbols **On/Off**. If **On**, symbols at or close to the subpage border are plotted

Symbol Quality

Specifies the plotting quality for the symbols

Symbol Blanking

Toggles blanking of an area around the symbols **On/Off**

Symbol Blanking Gap

Specifies the gap (in cm) to be left blank around each symbol. Only available if **Symbol Blanking** set to **On**

Symbol Colour

Specifies the colour to be used for the plotting symbols. Only available for **Symbol Table Mode** set to **Off** (individual mode)

Symbol Format

Specifies the format to be used for the symbols (if these are numbers). Only available for **Symbol Table Mode** set to **Off** (individual mode) and **Symbol Type** set to **Number**. Enter a FORTRAN style format specifier (e.g. I4, F6.3)

Symbol Height

Specifies the height of the plotting symbol in cm. Only available for **Symbol Table Mode** set to **Off** (individual mode)

Symbol Input Text List

Specifies a list of text strings to be used. Only available for **Symbol Table Mode** set to **Off** (individual mode) and **Symbol Type** set to **Text**. Enter a list of strings separated by forward slashes. If the number of strings specified does not match the number of values to be plotted only the first string in the list is used

Symbol Input Marker List

Specifies a list of markers to be used. Only available for **Symbol Table Mode** set to **Off** (individual mode) and **Symbol Type** set to **Marker**. Enter a list of numbers separated by forward slashes. Each number (from 0 to 28) is a numeric code for a given marker - see the correspondence number-symbol in the MAGICS Users Guide, ChapterXII - Symbol Plotting. If the number of markers specified does not match the number of values to be plotted only the first marker in the list is used

Symbol Min Table

Specifies a list of values which are the minima of ranges of values. Only available for **Symbol Table Mode** set to **On**. A different marker, number or text will be plotted for the values within each range. If ranges are not contiguous, nothing will be plotted for the values between ranges. The markers or text to be used are specified in **Symbol Marker Table** and **Symbol Text Table**

Symbol Max Table

Specifies a list of values which are the maxima of ranges of values. Only available for **Symbol Table Mode** set to **On**. A different marker, number or text will be plotted for the values within each range. If ranges are not contiguous, nothing will be plotted for the values between ranges. The markers or text to be used are specified in **Symbol Marker Table** and **Symbol Text Table**

Symbol Marker Table

Specifies the list of marker indices (numbers) which will be plotted for each of the ranges of values defined by **Symbol Min Table** and **Symbol Max Table**. Only available for **Symbol Table Mode** set to **On** and **Symbol Table** set to **Marker**. Enter a list of numbers separated by a forward slash. You must specify as many indices as the number of ranges of values you have defined, even if you want to use the same marker throughout (repeat the required number of times)

Symbol Text Table

Specifies the list of text strings which will be plotted for each of the ranges of values defined by **Symbol Min Table** and **Symbol Max Table**. Only available for **Symbol Table Mode** set to **On** and **Symbol Table** set to **Text**. Enter a list of strings separated by a forward slash. You must specify as many strings as the number of ranges of values you have defined, even if you want to use the same string throughout (repeat the required number of times)

Symbol Colour Table

Specifies a list of colours to be applied to the selected markers. Specify as many colours as the number of ranges of values you have defined by **Symbol Min Table** and **Symbol Max Table**. On the colour help tool, click the colours in the desired order of appearance. New colours are added to the end of the list, hence be careful with misplacement of colours in a long list. To remove a colour from the list, click left again on it

Symbol Height Table

Specifies a list of heights (in cm) to be applied to the selected markers. Specify as many heights as the number of ranges of values you have defined by **Symbol Min Table** and **Symbol Max Table**. Enter a list of numbers separated by a forward slash

Symbol Wind Origin Marker

Toggles plotting of a marker at the origin of the wind arrow **On/Off**. Only available for **Symbol Type** set to **Wind**.

Symbol Distance Apart

Specifies the minimum distance in cm between the centre of any two plotting symbols

Actions on the Symbol Icon

The actions available for the Symbol icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

GRAPH



This is the visual definition used for the plotting of graphs. It is the translation of the **MAGICS** graph plotting directives into a Metview icon. The macro language equivalent `pgraph()`

Further details, not presented in this manual can be found in Chapter 10 ("Graph Plotting") of the **MAGICS** Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>.

The Graph Editor

The Graph editor contains parameters for the plotting specification of three types of graphs - line (curve), bar and area. As one of these types is chosen, parameters for the other types become unavailable.

Legend

Toggles the user defined legend **On** or **Off**.

Legend User Text

Type the text you want to go with the legend. Unavailable if **Legend** set to **Off**.

Legend Entry

Drop in an icon of a legend definition (see "Legend Entry" on page III- 231). Unavailable unless **Legend** is set to **On**

Graph Position Mode

Specifies the positioning mode of the symbols. Options are :

- **Axis** - applies to curves in x-axis, y-axis space
- **Geographic** - applies to curves in geographic space
- **Paper** - applies to curves specified in cm relative to the lower left corner of the subpage

Graph Type

Specifies the three graph plotting options :

- **Curve** - line charts. The lines to be drawn are described by their x,y values within a set of XY axis. You can plot the curve with a line and/or with marker symbols (see below)
- **Bar** - bar charts
- **Area** - line charts with shading of the area between two curves

Graph Curve Method

Specifies the curve plotting method - **Straight** or **Rounded** (plotted with smoothing). Not available for **Graph Type** set to **Bar**

Graph Line

Toggles the plotting of lines in the graph **On/Off**. Only available for **Graph Type** set to **Curve** or **Bar**

Graph Line Parameters

Comprising **Graph Line Colour**, **Graph Line Style** and **Graph Line Thickness**, these parameters specify the characteristics of the graph line to plot. Only available for **Graph Line** set to **On** and **Graph Type** set to **Curve**

Graph Blanking

Toggles blanking of an area around the curve **On/Off**. Only available for **Graph Line** set to **On** and **Graph Type** set to **Curve**

Graph Blanking Gap

Specifies the width in cm of the area to be blanked around the curve. Only available for **Graph Blanking** set to **On**

Graph Symbol

Toggles plotting of markers **On/Off**. Only available for **Graph Type** set to **Curve**

Graph Symbol Parameters

Comprising **Graph Symbol Height**, **Graph Symbol Colour** and **Graph Symbol Marker Index**, these parameters specify the characteristics of the symbols to be used in plotting. The **Graph Symbol Marker Index** is a number (from 0 to 20) which corresponds to a particular marker; see the correspondence number-symbol in Modifications and Additions to the MAGICS Reference Manual v1.2, chapter 14, appendix 5, page 5.1. Only available if **Graph Symbol** is **On**

Graph Missing Data Mode

Specifies the procedure to adopt for missing data values. Missing is understood to apply to values which are outside the current axis system. It is only available for **Graph Type** set to **Curve**. Options are :

- **Ignore** - only the parts of the curve for which data exists are plotted. No lines are drawn between the points preceding and following the missing values
- **Join** - plot the curve as if the missing values were not there. A line is drawn between the points preceding and following the missing values
- **Drop** - a line is drawn from the preceding point in the direction of the first missing point until it reaches the X axis. Then it is plotted along the X axis until the last missing value when it will be drawn towards the first point that follows the missing values

Graph Missing Data Parameters

Comprising **Graph Missing Data Colour**, **Graph Missing Data Style** and **Graph Missing Data Thickness**, these parameters specify the characteristics of the line used in the joining or dropping of missing data values (see above). Only available for **Graph Missing Data Mode** set to **Join** or **Drop**

Graph Bar Parameters

Comprising **Graph Bar Colour**, **Graph Bar Width**, **Graph Bar Line Style** and **Graph Bar Line Thickness**, these parameters specify the characteristics of the bars to be plotted. Only available for **Graph Type** set to **Bar**

Graph Shade

Toggles shading of bars or areas between curves **On/Off**. Not available for **Graph Type** set to **Curve**

Graph Shade Parameters

Comprising **Graph Shade Colour**, **Graph Shade Style**, **Graph Shade Density**, **Graph Shade Dot Size** and **Graph Shade Hatch Index**, these parameters specify the characteristics of the shading of bars or areas between curves. Not available for **Graph Type** set to **Curve**

Graph X Suppress Below

Ignore data whose x value is below the value specified for this parameter

Graph X Suppress Above

Ignore data whose x value is above the value specified for this parameter

Graph Date X Suppress Below

Ignore data whose date x value is below (sooner) the value specified for this parameter

Graph Date Y Suppress Above

Ignore data whose date y value is above (later) the value specified for this parameter

Graph Y Suppress Below

Ignore data whose y value is below the value specified for this parameter

Graph Y Suppress Above

Ignore data whose y value is above the value specified for this parameter

Hatch Patterns for Graphs

The following figure shows the hatching patterns available, specified with parameter **Graph Shade Hatch Index**.

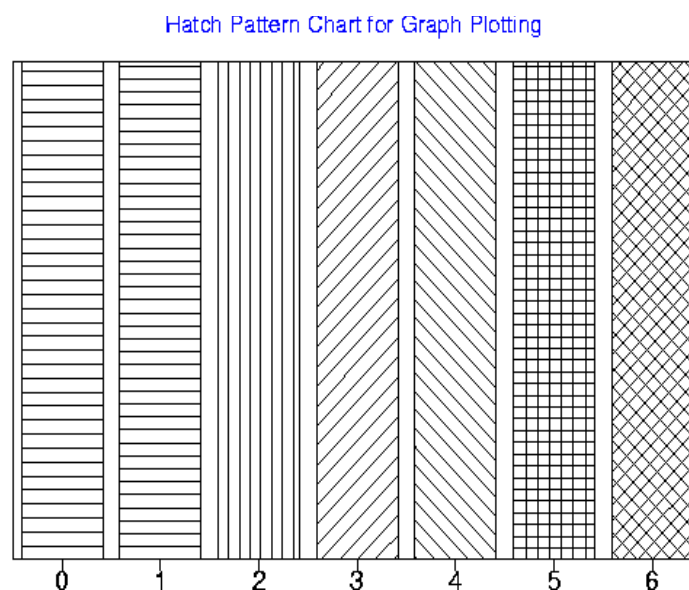


Figure III-7 - Hatch patterns for graphs

Actions on the Graph Icon

The actions available for the Graph icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

AXIS



This is the visual definition used for the plotting of axis. It is the translation of the MAGICS axis plotting directives into a Metview icon. The macro language equivalent is `paxis()`

Further details, not presented in this manual can be found in Chapter 9 ("Axis Plotting") of the MAGICS Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>.

The Axis Editor

Axis Type

Specifies the type of axis to be used. The choice made here makes available/unavailable blocks of parameters below :

- **Regular** - standard axis
- **Logarithmic** - logarithmic axis
- **Date** - time axis. Values entered in **Axis Min Value** and **Axis Max Value** below, must be specified as dates in the format `YYYYMMDD00` (e.g. 1997120900 for the 9th December 1997). This option makes available a number of parameters further below (**Axis Base Date** and following)
- **Position List** - axis for irregularly placed values. See **Axis Tick Position List** (page III-227) and **Axis Tick Label List** (page III-227)

Axis Min Value

Specifies the minimum value to appear on the axis (and to be plotted). Not available for **Axis Type** set to **Date**.

Axis Max Value

Specifies the maximum value to appear on the axis (and to be plotted). Not available for **Axis Type** set to **Date**.

Axis Date Min Value

Specifies the minimum value (start date) to appear on a date axis (and to be plotted). Enter a date in format `YYYY-MM-DD HH:mm:ss`. Only available for **Axis Type** set to **Date**.

Axis Date Max Value

Specifies the maximum value (end date) to appear on the axis (and to be plotted). Enter a date in format `YYYY-MM-DD HH:mm:ss`. Only available for **Axis Type** set to **Date**.

Axis Orientation

Toggles the orientation of the axis between **Horizontal** (x-axis) and **Vertical** (y-axis)

Axis Position

Specifies the placement of the axis. Options are **Top**, **Bottom**, **Left**, **Right** and **Positional**. If set to **Positional**, you can place the axis anywhere you want. You should enter the distance of the axis origin to the bottom left corner of the subpage in **Axis Line Position** (see below)

Axis Line

Toggles the axis line **On/Off** (visible or not)

Axis Line Parameters

Specify the **Colour** and **Thickness** of the axis lines

Axis Line Position

Specifies the distance from the bottom left corner of the subpage to the origin of the axis in cm. See **Axis Position** above

Axis Title

Toggles the axis title (printed along the axis) **On/Off** . If **Off**, the following Axis Title related parameters become unavailable

Axis Title Orientation

Specifies the title orientation. Options are **Parallel**, **Horizontal**, **Vertical**

Axis Title Parameters

Specify the **Colour**, **Height** and **Quality** of the axis title

Axis Title Text

Enter the text to be plotted as the axis title

Axis Tip Title

Toggles the axis tip title **On/Off**. The tip title sits at the extremity of the axis and is typically used for unit labeling

Axis Tip Title Parameters

Specify the **Orientation**, **Colour**, **Height** and **Quality** of the axis tip title

Axis Tip Title Text

Enter the text to be plotted as the axis tip title

Axis Tick

Toggles the axis tick marks **On/Off**

Axis Tick Parameters

Specify the **Colour**, **Size** and **Thickness** of the axis ticks

Axis Tick Interval

Specifies the tick spacing. Enter n to plot every n^{th} tick. Not available for **Axis Tick Positioning** set to **Logarithmic** or **Position List**

Axis Tick Label

Toggles labels on axis ticks **On/Off**

Axis Tick Label Parameters

Specify the **Orientation**, **Colour**, **Height**, **Quality** and **Position** of the axis tick labels

Axis Tick Label First / Axis Tick Label Last

Toggle the first and last axis tick labels **On/Off**

Axis Tick Label Type

Specifies the type of the axis tick label. This depends on what you have chosen for **Axis Type** (if set to **Position List** this parameter is unavailable). Options include :

- **Label List** - Use for **Axis Type** set to **Regular** or **Logarithmic**
- **Number** - Use for **Axis Type** set to **Regular** or **Logarithmic**
- **Latitude / Longitude** - Use for **Axis Type** set to **Geographic**

Axis Tick Label List

Specifies a list of labels to be used.

Axis Tick Label Format

Specifies the format of the labels. Use FORTRAN like format specifiers for numbers (e.g. I5, F4.1).
Not available for **Axis Type** set to **Position List**

Axis Tick Label Frequency

Specifies the label frequency. Enter *n* to plot every *n*th label. Not available for **Axis Type** set to **Position List** or **Axis Tick Label Type** set to **Label List**.

Axis Tick Position List

Specifies the list of tick positions to be used. Only available for **Axis Type** (page III-225) set to **Position List**

Axis Tick Label Units

Specifies the units for the placing of the labels (**cm** or **user** defined) Only available for **Axis Type** (page III-225) set to **Position List**

Axis Minor Tick

Toggles the axis minor tick **On/Off**. This parameter and all those below up to **Axis Grid** are unavailable for **Axis Type** set to **Position List**

Axis Minor Tick Count

Specifies the number of minor ticks between each axis tick

Axis Minor Tick Parameters

Specifies the **Colour**, **Size** and **Thickness** of the minor ticks

Axis Minor Tick Min Gap

Specifies the minimum gap between minor ticks in cm

Axis Date Type

Specifies which type of date your axis describes - **Years**, **Months**, **Days**, **Hours**.

Axis Hours Label

Toggles the presence of hour elements in date/time axis labels **On/Off**. Only available for **Axis Date Type** set to **Hours**.

Axis Hours Label Parameters

Specifies the **Height**, **Quality** and **Colour** of the hour labels

Axis Days Label

Specifies the type/presence of day elements in date/time axis labels. If set to **Days**, days of the week name are used, if set to **Number**, day number of the month is used, if set to **Both**, both types are used. If set to **Off**, day labels are turned off and the four **Axis Days Label** parameters that follow become unavailable

Axis Days Label Parameters

Specifies the **Height**, **Quality** and **Colour** of the day labels

Axis Days Label Composition

Specifies the number of characters (**One** - w, **Three** - WED or **Full** - WEDNESDAY), to be used for day labels

Axis Months Label

Toggles the presence of month elements in date/time axis labels **On/Off**

Axis Months Label Parameters

Specifies the **Height**, **Quality** and **Colour** of the month labels

Axis Months Label Composition

Specifies the number of characters (**One** - D, **Three** - DEC or **Full** - DECEMBER), to be used for month labels

Axis Years Label

Toggles the presence of year elements in date/time axis labels **On/Off**

Axis Years Label Parameters

Specifies the **Height**, **Quality** and **Colour** of the year labels

Axis Grid

Toggles an axis grid **On/Off**

Axis Grid Parameters

Specifies the **Colour**, **Thickness** and **Line Style** of the axis grid

Axis Date Grid Positioning

Specifies the type of positioning for a date axis grid - **Automatic** and **Positional**

Axis Date Grid Position List

Specifies the grid positions for date axis

Actions on the Axis Icon

The actions available for the Axis icon are *Visualise*, *Edit*, *Duplicate* and *Delete*.

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

IMAGE TABLE



This is the visual definition used for the plotting of satellite images. The macro language equivalent is `pmimage()`.

Further details, not presented in this manual can be found in Chapter 13 ("Satellite Image Plotting") of the MAGICS Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>.

The Image Table Editor

Image Min Value

Specifies the lowest pixel value to be plotted.

Image Max Value

Specifies the highest pixel value to be plotted.

Image Colour Table Type

Specifies whether the colour table should be **Computed** automatically between two colours, or specified completely by the **User**.

Image Min Level Colour

Specifies the colour assigned to the lowest pixel value to be plotted. Only available if **Image Colour Table Type** is set to **Computed**.

Image Max Level Colour

Specifies the colour assigned to the highest pixel value to be plotted. Only available if **Image Colour Table Type** is set to **Computed**.

Image Colour Table Creation Mode

Only available if **Image Colour Table Type** is set to **Computed**. Specifies the method for computing the colour table for plotting the image

- **Default** - Even distribution of colours over the full 128-255 range of pixels.
- **Normal** - Even distribution of the colours according to the range of pixel values.
- **Equidistant** - Even distribution of colours according to the number of pixel values in each shading band.

Image Colour Direction

Toggles the direction of colour selection between **Image Min Level Colour** and **Image Maximum Level Colour** in the 360° Hue scale between **Clockwise** and **Anti Clockwise**.

Image Colour Table

Only available if **Image Colour Table Type** is set to **User**. Allows a user-defined colour table to be used.

Image Background Colour

Specifies the colour of the area on the map that is not part of the satellite image.

Image Pixel Selection Frequency

Number of pixels/centimetre to be plotted (measured along the subpage frame)

Image Legend

Toggles plotting of the image legend **On/Off**.

Image Legend Title

Toggles plotting of the image legend title **On/Off**.

Image Legend Title Text

Specifies the text to appear as legend text

Image Legend Text Quality

Specifies the quality of the legend text

Image Legend Text Colour

Specifies the colour of the legend text

Image Legend Border

Toggles the border of the legend **On/Off**.

Image Legend Border Parameters

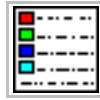
Comprising **Image Legend Border Line Style**, **Image Legend Border Colour**, **Image Legend Border Thickness**, and **Image Legend Border Blanking**, these parameters specify the characteristics of the legend border to be plotted.

Actions on the Image Table Icon

The actions available for the Image Table icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

LEGEND ENTRY



This is the visual definition used for the specification of legend text. The macro language equivalent is `legendentry()`

A legend entry is composed of three parts - a symbol, automatic text and user text. The symbol may be a shading pattern or a portion of a contour line. The automatic text normally consists of the numerical values associated with the symbol. User text is defined by you and may be combined with automatic text or you may choose to use any of the two in isolation.

This icon is only used by dropping in icon fields of other visual definition icons. Those that accept/generate legends are Contour, Graph and Wind Plot

Further details, not presented in this manual can be found in Chapter 15 ("Legend Plotting") of the MAGICS Users Guide in <http://www.ecmwf.int/publications/manuals/magics/manuals/index.html>.

The Legend Entry Editor

Legend Sequence Number

Specifies the order number of the first legend entry to be plotted

Legend Display Type

Toggles the type of shaded legend between **Disjoint** and **Continuous**

Legend Text Composition

Specifies the legend text plotting mode. Options are **Both** (use both user and automatic text), **Automatic Text Only** and **User Text Only**

Legend Text Format

Specifies the format of the legend's automatic text. Unavailable for **Legend Text Composition** set to **User Text Only**. Use FORTRAN style format specifiers

Legend Text Colour

Specifies the colour of the legend text

Legend User Text N

Enter the user text to be printed at the side of each legend symbol element. Unavailable for **Legend Text Composition** set to **Automatic Text Only**. This text may be used instead of the automatic text (**Legend Text Composition** set to **User Text Only**) or together with the automatic text (**Legend Text Composition** set to **Both**), e.g. to specify the units of the numbers generated by the automatic text

Legend Automatic Text Extrema

Specifies whether to include the maximum and/or minimum of the field in the legend. Options are **Both**, **Min**, **Max** and **None**. Applies to contour legends only

Legend Level List Entries

Use only if the Legend Entry icon is to be dropped in a Contour icon and if within this Contour icon the parameter **Contour Level Selection Type** is set to **Level List**. Specifies how many of the shading bands specified by the user are to be plotted. Options are :

- **All** - include all shading bands even if not actually plotted
- **As Plotted** - include only the shading bands which are actually plotted

Actions on the Legend Entry Icon

The actions available for the Legend Entry icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in icon fields of editors of other icons that accept it : Contour, Graph and Wind Plot

IMPORT PLOT



This is the visual definition used to control import of graphics files into a display window. The macro language equivalent is `pimport()`.

The Import Plot Editor

Import File Resolution

Specifies the resolution to use in the display of the imported graphics file.

Import File Alignment

Specifies the positioning of the import file in the page.

Actions on the Import Plot Icon

The actions available for the Import Plot icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

DRAWING PRIORITY



This is the visual definition used to control the plotting sequence of visual definitions (or parts thereof) such as coastlines, contour lines, contour shades, etc.. The macro language equivalent is `drawing_priority()`.

The Drawing Priority Editor

This editor contains a list of other visual definitions or of other visual definition components. Each is associated with an option list which specifies its *plotting rank* or *drawing priority*.

This plotting rank is either a rank from 100 (first) to 900 (last) or a simple binary first/last option. This allows you to control which visual definition (or component of) is plotted in which sequence.

The most obvious application of this icon is when users want to mask ocean/land areas from their plots - all you need to do is use a Coastline visual definition using ocean/land shade and then set the drawing priority of the ocean/land shade to last.

In this way the ocean/land shade is plotted last thereby overlaying and masking out all other plotting elements (contour lines, observation plots, wind arrows, ...). Elements not listed on this icon (e.g. grid lines) will always be plotted in their default position.

Actions on the Drawing Priority Icon

The actions available for the Drawing Priority icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

OVERLAY CONTROL



This is the visual definition used to control the overlaying of data in the same page. Full details about its working are given in "Data overlay control in view icons" on page I- 88 and "Data overlay rules and overlay controls" on page I- 116ff.

The macro language equivalent is `overlay_control()`.

The Overlay Control Editor

Overlay Mode

Specifies the overlay mode to be used - always/never overlay data units or check the settings. The settings are specified in the following input parameters

Overlay Date

Specifies whether to overlay data units based on forecast verification date

Overlay Time

Specifies whether to overlay data units based on forecast verification time

Overlay Level

Specifies whether to overlay data units based on their level (model or pressure)

Overlay Area

Specifies whether to overlay data units based on the area covered by the data

Obs Grouping Period

Specifies the number of hours over which to group observations

Obs Grouping Period Start Min

Specifies the number of minutes past the hour that the grouping period starts

Actions on the Overlay Control Icon

The actions available for the Overlay Control icon are *Edit*, *Duplicate* and *Delete*

To apply this visual definition, drop in a display window following or simultaneously with, an icon containing data for visualisation. See more details in "Visual Definitions - Plotting Control" on page I- 64 and "Visual Definition Icon Drops" on page I- 119.

