NORTH WEST SHELF
JOINT ENVIRONMENTAL
MANAGEMENT STUDY

Management strategy evaluations for multiple use management of Australia's North West Shelf – visualisation software user guide and outline

NWSJEMS

TECHNICAL REPORT No. 17

• B. Hatfield • L. Thomas • R. Scott

June 2006

CSIRO

Government of Western Australia
Department of Environment

National Library of Australia Cataloguing-in-Publication data:

Hatfield, B. (Brian), 1961- .
Management strategy evaluations for multiple use management of Australia's North West Shelf : visualisation software and user guide.

Bibliography.
Includes index.
ISBN 1 921061 83 9 (pbk.).

1. Marine resources - Western Australia - North West Shelf - Management - Data processing. 2. Marine resources conservation - Western Australia - North West Shelf - Data processing. 3. Environmental management - Western Australia - North West Shelf - Data processing. I. Thomas, L. (Linda), 1975- . II. Scott, R. (Roger), 1969- . III. CSIRO. Marine and Atmospheric Research. North West Shelf Joint Environmental Management Study. IV. Western Australia. V. Title. (Series : Technical report (CSIRO. Marine and Atmospheric Research. North West Shelf Joint Environmental Management Study) ; no. 17).

333.9164099413

Hatfield, B. (Brian), 1961- .
Management strategy evaluations for multiple use management of Australia's North West Shelf : visualisation software and user guide.

Bibliography.
Includes index.
ISBN 1 921061 84 7 (CD-ROM).

1. Marine resources - Western Australia - North West Shelf - Management - Data processing. 2. Marine resources conservation - Western Australia - North West Shelf - Data processing. 3. Environmental management - Western Australia - North West Shelf - Data processing. I. Thomas, L. (Linda), 1975- . II. Scott, R. (Roger), 1969- . III. CSIRO. Marine and Atmospheric Research. North West Shelf Joint Environmental Management Study. IV. Western Australia. V. Title. (Series : Technical report (CSIRO. Marine and Atmospheric Research. North West Shelf Joint Environmental Management Study) ; no. 17).

333.9164099413

Hatfield, B. (Brian), 1961- .
Management strategy evaluations for multiple use management of Australia's North West Shelf : visualisation software and user guide.

Bibliography.
Includes index.
ISBN 1 921061 85 5 (pdf).

1. Marine resources - Western Australia - North West Shelf - Management - Data processing. 2. Marine resources conservation - Western Australia - North West Shelf - Data processing. 3. Environmental management - Western Australia - North West Shelf - Data processing. I. Thomas, L. (Linda), 1975- . II. Scott, R. (Roger), 1969- . III. CSIRO. Marine and Atmospheric Research. North West Shelf Joint Environmental Management Study. IV. Western Australia. V. Title. (Series : Technical report (CSIRO. Marine and Atmospheric Research. North West Shelf Joint Environmental Management Study) ; no. 17).

333.9164099413

NORTH WEST SHELF JOINT ENVIRONMENTAL MANAGEMENT STUDY

Final report

North West Shelf Joint Environmental Management Study Final Report.

List of technical reports

NWSJEMS Technical Report No. 1
Review of research and data relevant to marine environmental management of Australia's North West Shelf.
A. Heyward, A. Revill and C. Sherwood

NWSJEMS Technical Report No. 2
Bibliography of research and data relevant to marine environmental management of Australia's North West Shelf.
P. Jernakoff, L. Scott, A. Heyward, A. Revill and C. Sherwood

NWSJEMS Technical Report No. 3
Summary of international conventions, Commonwealth and State legislation and other instruments affecting marine resource allocation, use, conservation and environmental protection on the North West Shelf of Australia.
D. Gordon

NWSJEMS Technical Report No. 4
Information access and inquiry.
P. Brodie and M. Fuller

NWSJEMS Technical Report No. 5
Data warehouse and metadata holdings relevant to Australia's North West Shelf.
P. Brodie, M. Fuller, T. Rees and L. Wilkes

NWSJEMS Technical Report No. 6
Modelling circulation and connectivity on Australia's North West Shelf.
S. Condie, J. Andrewartha, J. Mansbridge and J. Waring

NWSJEMS Technical Report No. 7
Modelling suspended sediment transport on Australia's North West Shelf.
N. Margvelashvili, J. Andrewartha, S. Condie, M. Herzfeld, J. Parslow, P. Sakov and J. Waring

NWSJEMS Technical Report No. 8
Biogeochemical modelling on Australia's North West Shelf.
M. Herzfeld, J. Parslow, P. Sakov and J. Andrewartha

NWSJEMS Technical Report No. 9
Trophic webs and modelling of Australia's North West Shelf.
C. Bulman

NWSJEMS Technical Report No. 10
The spatial distribution of commercial fishery production on Australia's North West Shelf.
F. Althaus, K. Woolley, X. He, P. Stephenson and R. Little

NWSJEMS Technical Report No. 11
Benthic habitat dynamics and models on Australia's North West Shelf.
E. Fulton, B. Hatfield, F. Althaus and K. Sainsbury

NWSJEMS Technical Report No. 12
Ecosystem characterisation of Australia's North West Shelf.
V. Lyne, M. Fuller, P. Last, A. Butler, M. Martin and R. Scott

NWSJEMS Technical Report No. 13
Contaminants on Australia's North West Shelf: sources, impacts, pathways and effects.
C. Fandry, A. Revill, K. Wenziker, K. McAlpine, S. Apte, R. Masini and K. Hillman

NWSJEMS Technical Report No. 14
Management strategy evaluation results and discussion for Australia's North West Shelf.
R. Little, E. Fulton, R. Gray, D. Hayes, V. Lyne, R. Scott, K. Sainsbury and D. McDonald

NWSJEMS Technical Report No. 15
Management strategy evaluation specification for Australia's North West Shelf.
E. Fulton, K. Sainsbury, D. Hayes, V. Lyne, R. Little, M. Fuller, S. Condie, R. Gray, R. Scott, H. Webb, B. Hatfield, M. Martin, and D. McDonald

NWSJEMS Technical Report No. 16
Ecosystem model specification within an agent based framework.
R. Gray, E. Fulton, R. Little and R. Scott

NWSJEMS Technical Report No. 17
Management strategy evaluations for multiple use management of Australia's North West Shelf – Visualisation software and user guide.
B. Hatfield, L. Thomas and R. Scott

NWSJEMS Technical Report No. 18
Background quality for coastal marine waters of the North West Shelf, Western Australia.
K. Wenziker, K. McAlpine, S. Apte, R.Masini

CONTENTS

ACRONYMS

ACOM	Australian Community Ocean Model
AFMA	Australian Fisheries Management Authority
AFZ	Australian Fishing Zone
AGSO	Australian Geological Survey Organisation now Geoscience Australia
AHC	Australian Heritage Commission
AIMS	Australian Institute of Marine Science
AMSA	Australian Maritime Safety Authority
ANCA	Australian Nature Conservation Agency
ANZECC	Australian and New Zealand Environment and Conservation Council
ANZLIC	Australian and New Zealand Land Information Council
APPEA	Australian Petroleum, Production and Exploration Association
AQIA	Australian Quarantine Inspection Service
ARMCANZ	Agricultural Resources Management council of Australia and New Zealand
ASIC	Australian Seafood Industry Council
ASDD	Australian Spatial Data Directory
CAAB	Codes for Australian Aquatic Biota
CAES	Catch and Effort Statistics
CALM	Department of Conservation and Land Management (WA Government)
CAMBA	China Australia Migratory Birds Agreement
CDF	Common data format
CITIES	Convention on International Trade in Endangered Species
CTD	conductivity-temperature-depth
CMAR	CSIRO Marine and Atmospheric Research
CMR	CSIRO Marine Research
COAG	Council of Australian Governments
ConnIe	Connectivity Interface
CPUE	Catch per unit effort
CSIRO	Commonwealth Science and Industrial Research Organisation
DCA	detrended correspondence analysis
DIC	Dissolved inorganic carbon
DISR	Department of Industry, Science and Resources (Commonwealth)
DEP	Department of Environmental Protection (WA Government)
DOM	Dissolved organic matter
DPIE	Department of Primary Industries and Energy
DRD	Department of Resources Development (WA Government)
EA	Environment Australia
EEZ	Exclusive Economic Zone
EIA	Environmental Impact Assessment
EPA	Environmental Protection Agency
EPP	Environmental Protection Policy
ENSO	El Nino Southern Oscillation
EQC	Environmental Quality Criteria (Western Australia)
EQO	Environmental Quality Objective (Western Australia)
ESD	Ecologically Sustainable Development
FRDC	Fisheries Research and Development Corporation
FRMA	Fish Resources Management Act
GA	Geoscience Australia formerly AGSO
GESAMP	Joint Group of Experts on Scientific Aspects of Environmental Protection
GIS	Geographic Information System
ICESD	Intergovernmental Committee on Ecologically Sustainable Development
ICS	International Chamber of Shipping
IOC	International Oceanographic Commission
IGAE	Intergovernmental Agreement on the Environment
ICOMOS	International Council for Monuments and Sites

IMO	International Maritime Organisation
IPCC	Intergovernmental Panel on Climate Change
IUNC	International Union for Conservation of Nature and Natural Resources
IWC	International Whaling Commission
JAMBA	Japan Australian Migratory Birds Agreement
LNG	Liquified natural gas
MarLIN	Marine Laboratories Information Network
MARPOL	International Convention for the Prevention of Pollution from Ships
MECO	Model of Estuaries and Coastal Oceans
MOU	Memorandum of Understanding
MPAs	Marine Protected Areas
MEMS	Marine Environmental Management Study
MSE	Management Strategy Evaluation
NCEP - NCAR	National Centre for Environmental Prediction – National Centre for Atmospheric Research
NEPC	National Environmental Protection Council
NEPM	National Environment Protection Measures
NGOs	Non government organisations
NRSMPA	National Representative System of Marine Protected Areas
NWQMS	National Water Quality Management Strategy
NWS	North West Shelf
NWSJEMS	North West Shelf Joint Environmental Management Study
NWSMEMS	North West Shelf Marine Environmental Management Study
ICIMF	Oil Company International Marine Forum
OCS	Offshore Constitutional Settlement
PFW	Produced formation water
P(SL)A	Petroleum (Submerged Lands) Act
PSU	Practical salinity units
SeaWiFS	Sea-viewing Wide Field-of-view Sensor
SOI	Southern Oscillation Index
SMCWS	Southern Metropolitan Coastal Waters Study (Western Australia)
TBT	Tributyl Tin
UNCED	United Nations Conference on Environment and Development
UNCLOS	United Nations Convention of the Law of the Sea
UNEP	United Nations Environment Program
UNESCO	United Nations Environment, Social and Cultural Organisation
UNFCCC	United Nations Framework Convention on Climate Change
WADEP	Western Australian Department of Environmental Protection
WADME	Western Australian Department of Minerals and Energy
WAEPA	Western Australian Environmental Protection Authority
WALIS	Western Australian Land Information System
WAPC	Western Australian Planning Commission
WHC	World Heritage Commission
WOD	World Ocean Database
www	world wide web

TECHNICAL SUMMARY

As its name suggests, management strategy evaluation (MSE) provides the opportunity for managers to evaluate the performance of alternative natural resource management strategies by comparison among indicators of social, economic and environmental performance. One of the primary difficulties faced when comparing these indicators is the sheer volume of data which needs to be assimilated. In the case of the North West Shelf Joint Environmental Management Study (NWSJEMS) the combination of three operating models, three development scenarios and three management strategies offers a total of twenty seven possible alternatives for evaluation. For the NWSJEMS an agent based computer program, *InVitro*, was used to model the impact of future development in the North West Shelf. As *InVitro* has stochastic components it was necessary to have multiple instances of model runs using the same initial model parameters, a process which creates enormous quantities of data.

Two software packages were developed for dealing with NWSJEMS data. The first, *ViewNWS*, is specifically tailored for use by environmental managers needing to have an overall comparison of alternative management outcomes. It uses pre-packaged, processed data produced by *InVitro* and has a straightforward graphical user interface (GUI). The second software package, the *NWS Technical User Interface*, is targeted at modellers and programmers. It allows the user to visualise raw model output to allow rapid model and parameter changes.

1. INTRODUCTION

The computer software developed for application of the management strategy evaluation (MSE) framework in the NWSJEMS is made up of three components. The first software component supports the modelling of the system and completion of the MSE. This component is known as *InVitro* (described in the companion ecosystem model specification report Gray et al. 2006). This is enhanced with the second component, the NWS Technical User Interface, which provides scientific diagnostic tools for model development and improvement. The third component is known as *ViewNWS* and this provides interrogation and visualisation tools for examination of the MSE results.

This technical report provides a user guide for the visualisation software, *ViewNWS*, which is aimed at the general user (section 2). The programming documentation (section 3) and *NWS Technical User Interface* (section 4) gives the reader the chance to understand the structure of the software.

2. VIEWNWS: DATA VISUALISATION PACKAGE

2.1 Program description

ViewNWS: Data Visualisation Package is designed to enable environmental managers to quickly assess data produced in the NWS Management Strategy Evaluation (MSE) Project, identifying areas of interest and examine the MSE results.

ViewNWS: Data Visualisation Package provides two main functions.
These are to:

- provide a visualisation platform for GIS files; and
- provide a visual indication of the performance over time of several environmental indicators.

2.2 Program implementation

ViewNWS is coded in *Visual Basic Version 6*. The map displays incorporate *MapObjects*® (Esri 1999) Active X components which are embedded within the *Visual Basic* program. Extensive use is made of *Gigasoft ProEssentials ActiveX* charting components to display data within the program.

2.3 Data required

The program supports the following formats:

For map layers:

- Arc shape files (*.shp); and
- Arc INFO coverage files (*.adf)

For image layers:

- Geo-referenced image file (*.bil);
- Arc shape files (*.shp); and
- Arc INFO coverage files (*.adf).

The program also allows for the importation of non-spatial data from ".pt" files. These files are native to the *InVitro* agent based model.

2.4 Sources of data

The data shown in this document has been sourced from the following agencies:

- North West Shelf coastline – Geospatial and Earth Monitoring Division, Geoscience Australia;
- North West Shelf satellite image – Landsat, ACRES, Geoscience Australia; and
- Model output data, CSIRO Marine and Atmospheric Research, Hobart.

2.5 Setting up ViewNWS: Data Visualisation Package

See the README.txt file on the CD for instructions on how to install *ViewNWS* on your computer. Please note that it is essential to copy the "data" directory from the CD to the program directory (usually c:\program files\viewNWS).

2.6 Using the North West Shelf Data Visualisation Package

2.6.1 Launching the North West Shelf Data Visualisation Package

After installing the *North West Shelf Data Visualisation Package* software, open the program by accessing it from the start menu.

On the **start** menu, click **All Programs**, point to **ViewNWS**, click on **ViewNWS** (figure 2.6.1).

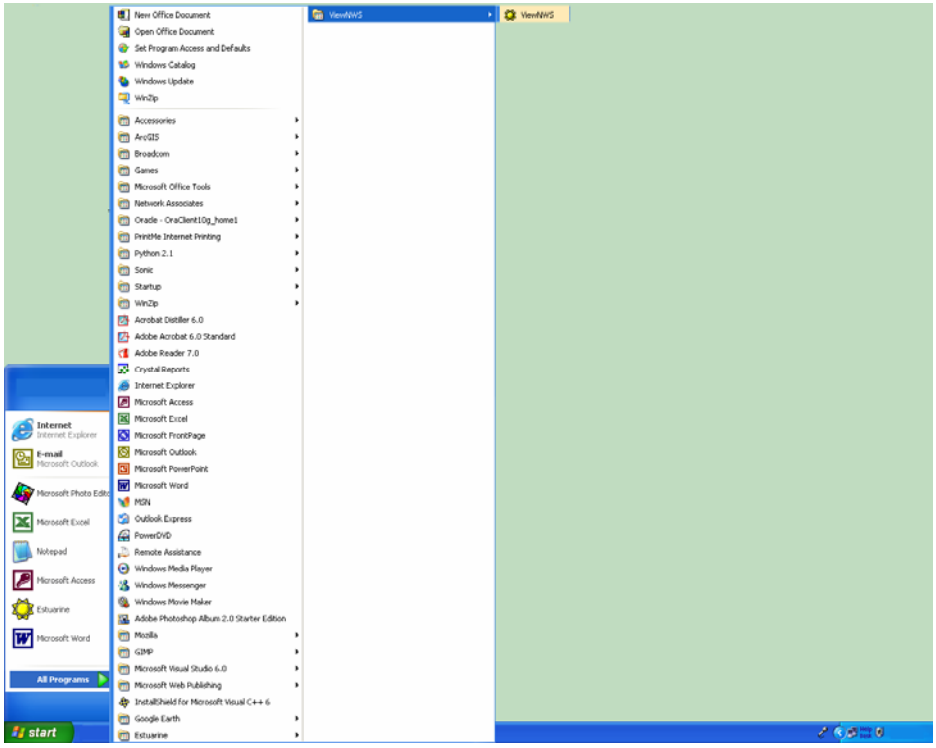


Figure 2.6.1: Launching the North West Shelf Data Visualisation Package from the start menu.

After the initial splash screen, an open map screen will be presented (figure 2.6.2).

From here any of the following can be selected by bringing the desired window to the front or by clicking on the corresponding buttons:

- launch the scenario chooser;
- display data on a map of the North West Shelf;
- display data on a satellite image underlay of the North West Shelf;
- import external data; or
- view a *PowerPoint* presentation of the North West Shelf area

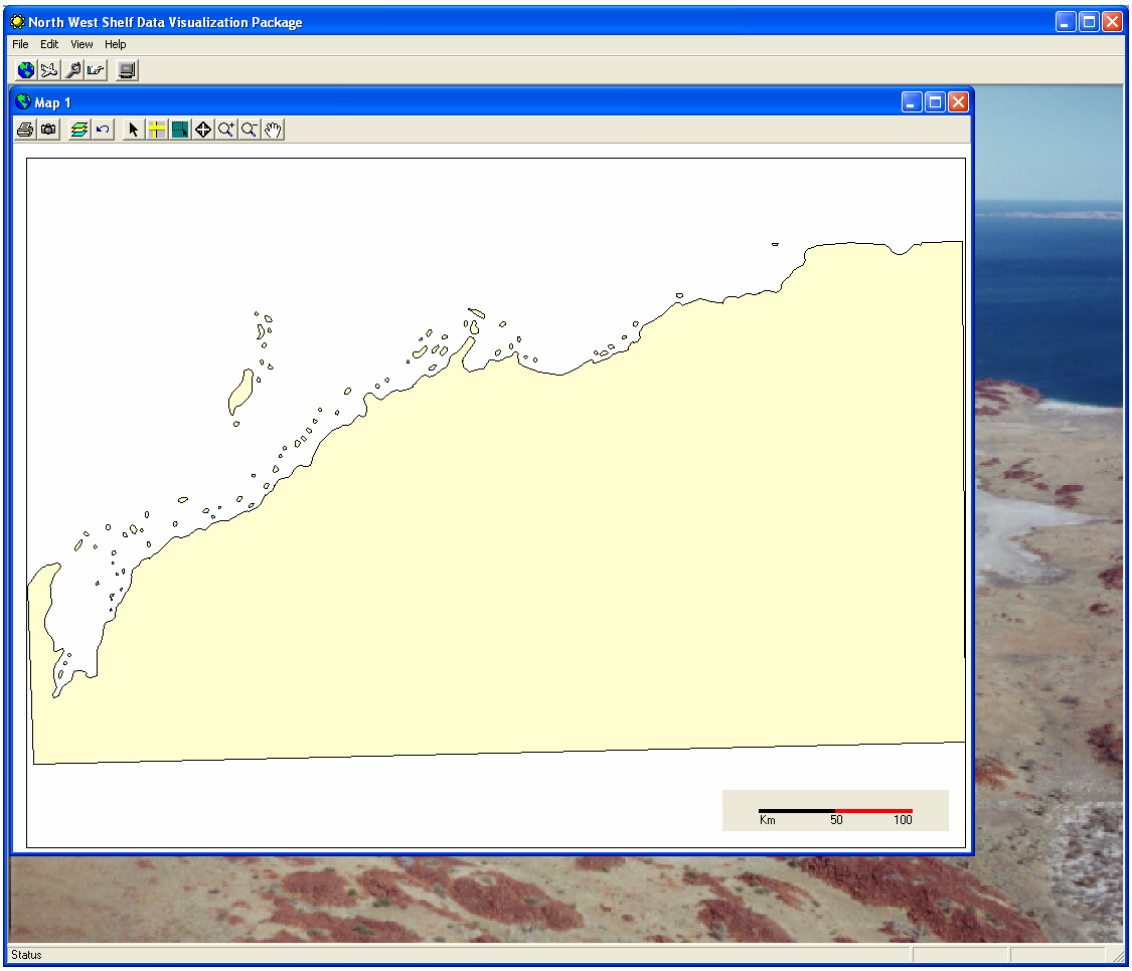


Figure 2.6.2: The main program screen.

2.6.2 Exiting the North West Shelf Data Visualisation Package

To exit *ViewNWS* select **Exit**, from the file drop down menu (figure 2.6.3).

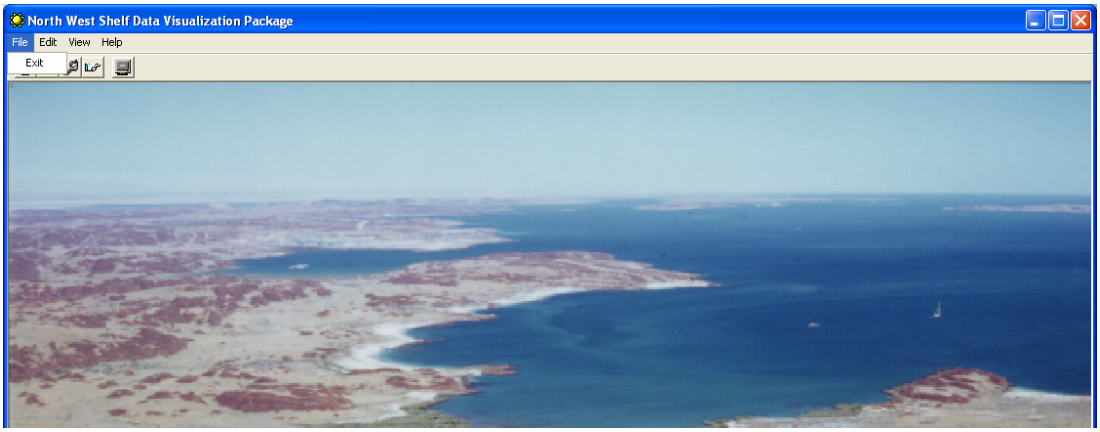


Figure 2.6.3: Exiting the program.

2.6.3 The two layers

ViewNWS provides a platform for viewing GIS files. These can be displayed in two ways. The first is a map layer which displays the data on a GIS shape file. The second is the image layer which displays data on a geo-referenced image file.

Both the map and image layers have common tool buttons which are displayed on the active view.

To change between two maximised open views (figure 2.6.4), click on the globe  that appears beside the **File** drop down menu and select next. Alternatively the short cut keys **Ctrl** + **F6** can be used.

Other icons can appear where the globe is depending on what application is open in *ViewNWS*. A spanner  will appear when the MSE screen is open and traffic lights  will appear when the indicator screen is open and there are other applications open.

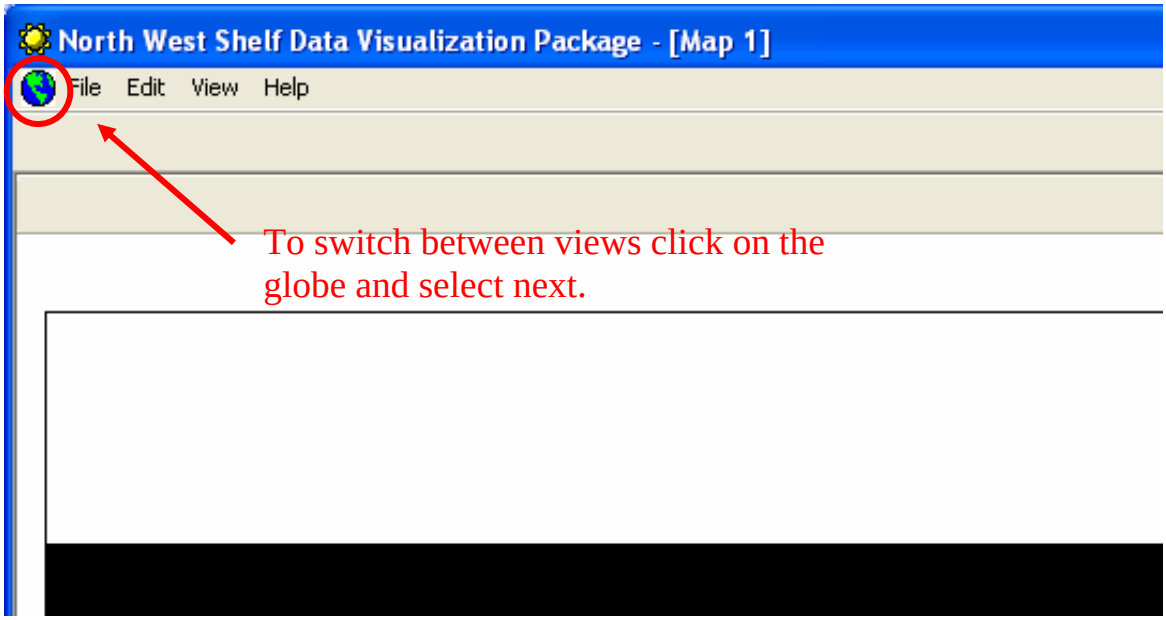


Figure 2.6.4: Switch between views.

2.6.4 Launching an image layer

To view data on the geo-referenced image layer click on **image layer**  to launch a blank image layer (figure 2.6.5).

The map has a geo-referenced image file (*.bil) for its underlying map file. This image file can be changed in the setup screen. See the section on creating set up parameters for instructions on how to do this.

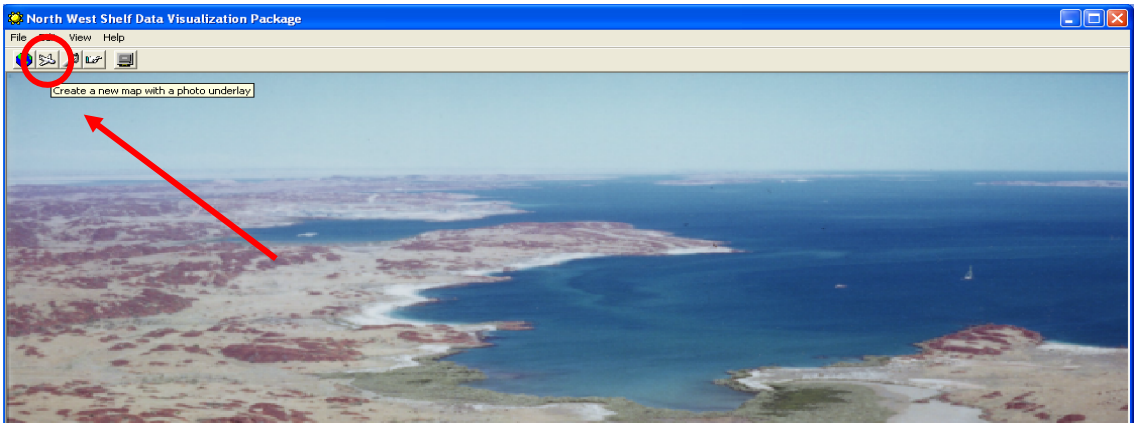


Figure 2.6.5: Launching an image layer.

By pressing **image layer**  a geo-referenced image of the study area will be presented (figure 2.6.6).

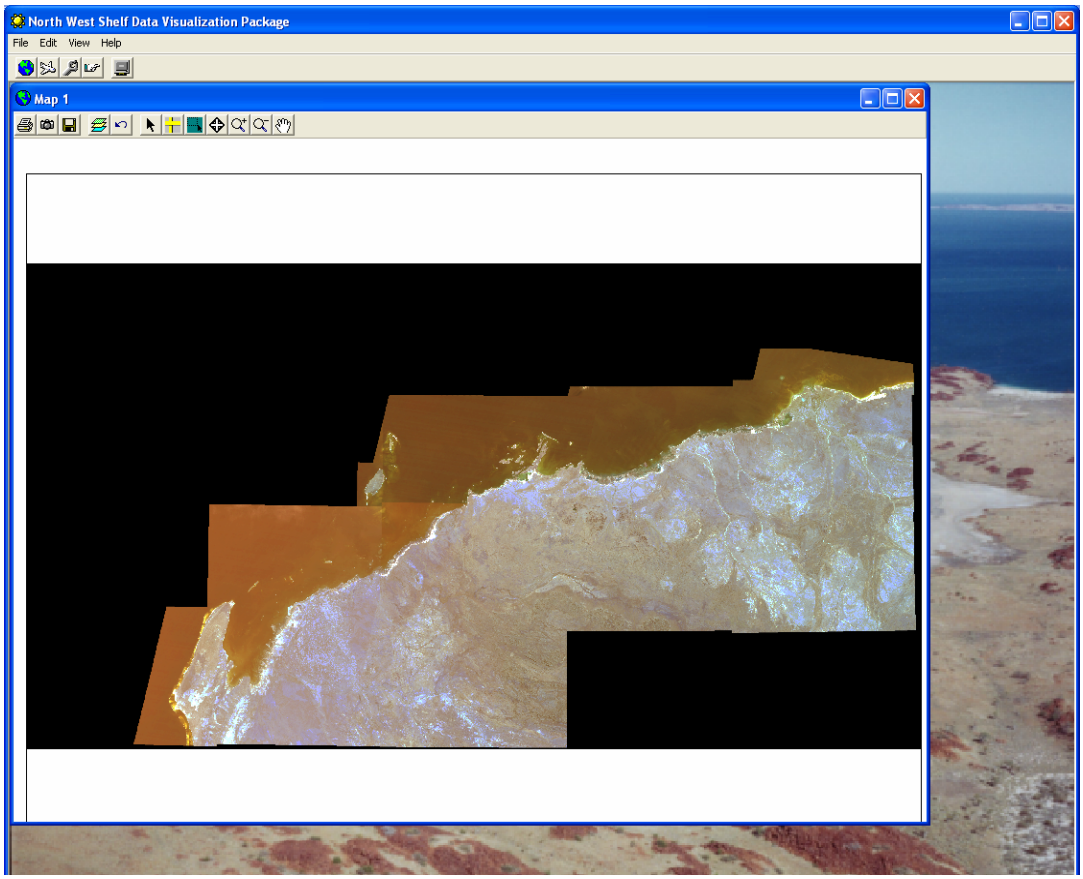


Figure 2.6.6: The initial geo-referenced map layer.

2.6.5 Launching a map layer

To view data on the map layer, click on **map layer**  to launch a blank map screen (figure 2.6.7).

The map layer screen is virtually identical to the image layer screen in functionality. It has a shape file for its underlying map file.

Figure 2.6.7: Launching a map layer.

Press **map layer**  to get a blank map screen (figure 2.6.8).

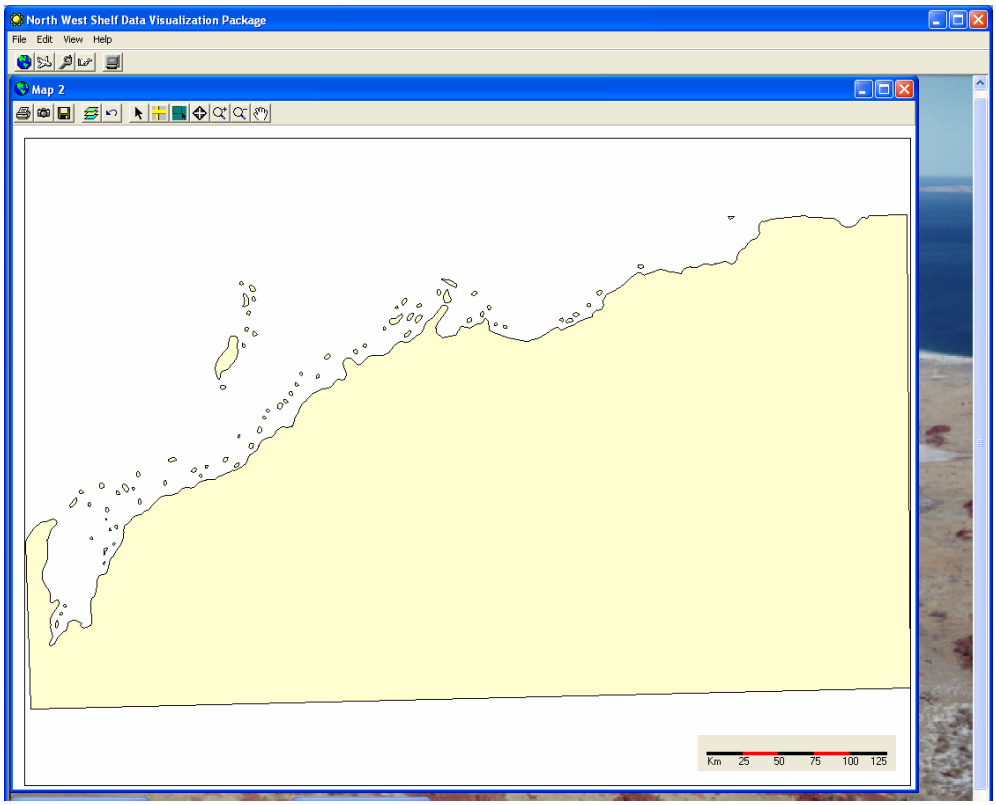


Figure 2.6.8: The initial map screen.

2.6.6 The tools

On each of the views there are tool buttons along the top of the window. These tool buttons allow the user to print, copy, add map layer, remove map layer, point, measure between points, select an area, zoom and pan, edit the map legend, show/hide onscreen controls and start/stop display (figure 2.6.9).

Table 2.6.1 gives a list of the buttons and their functions.

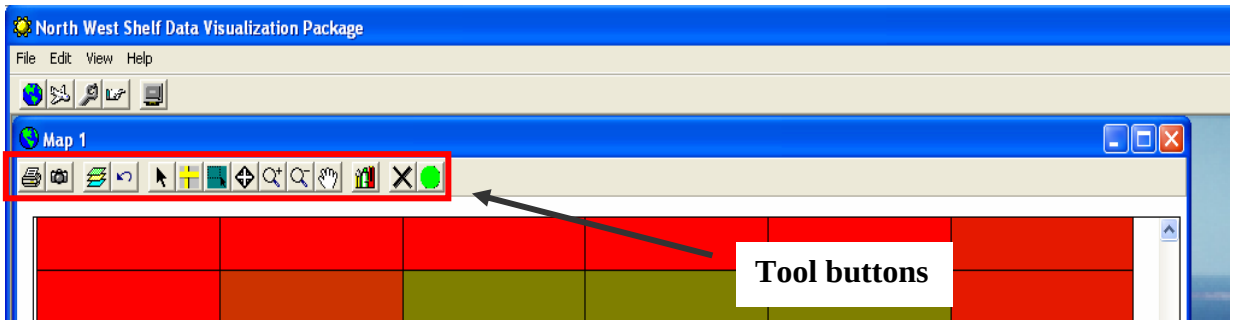


Figure 2.6.9: Tool buttons on the map window.

Table 2.6.1: Button functions.

Button	Button Name	Function
	Print	Print map
	Copy	Copy map to clipboard
	Add map layer	Add a map layer – multiple layers can be added
	Remove map layer	Remove a map layer – most recent first
	Point	Point to map
	Measure	Measure distance on map
	Select study area	Select a region
	Restore to normal size	Zoom to full extent
	Zoom in	Zoom in
	Zoom out	Zoom out
	Pan	Pan
	Edit map legend	Make changes to the map legend
	Show/hide on screen controls	Hides or shows the on screen controls
	Start/stop display	Starts and stops the display on the map.

NOTE: The following figures show the tool buttons as used in both the map layer and image layer. The tool buttons have the same function in both layers.

Adding and removing layers

To add a layer click on add **map layer**  . An open file dialogue box will be opened. Click on the **file** to be used and then click on **Open** (figure 2.6.10).

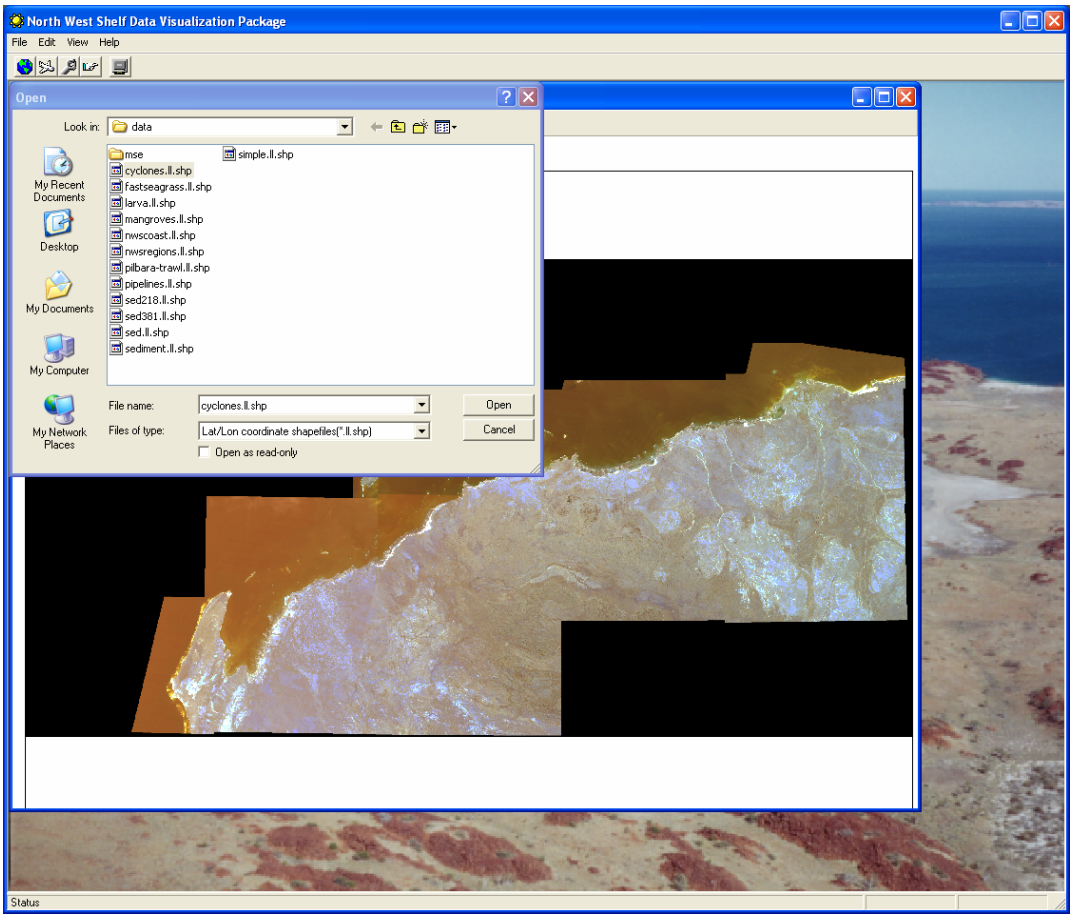


Figure 2.6.10: Add layer open file dialog box.

The map layer will then be displayed (figure 2.6.11). Repeat for any additional layers required.

To remove the map layer press **remove map layer**  .

In some instances the data being displayed has a time stamp. A box with three options will be presented (figure 2.6.12).

The first option allows all of the data to be displayed at once.

The second option allows a range of times to be selected and displayed from the on screen controls.

The final option allows the data to be displayed as a continuous loop, again this is controlled from the screen controls.

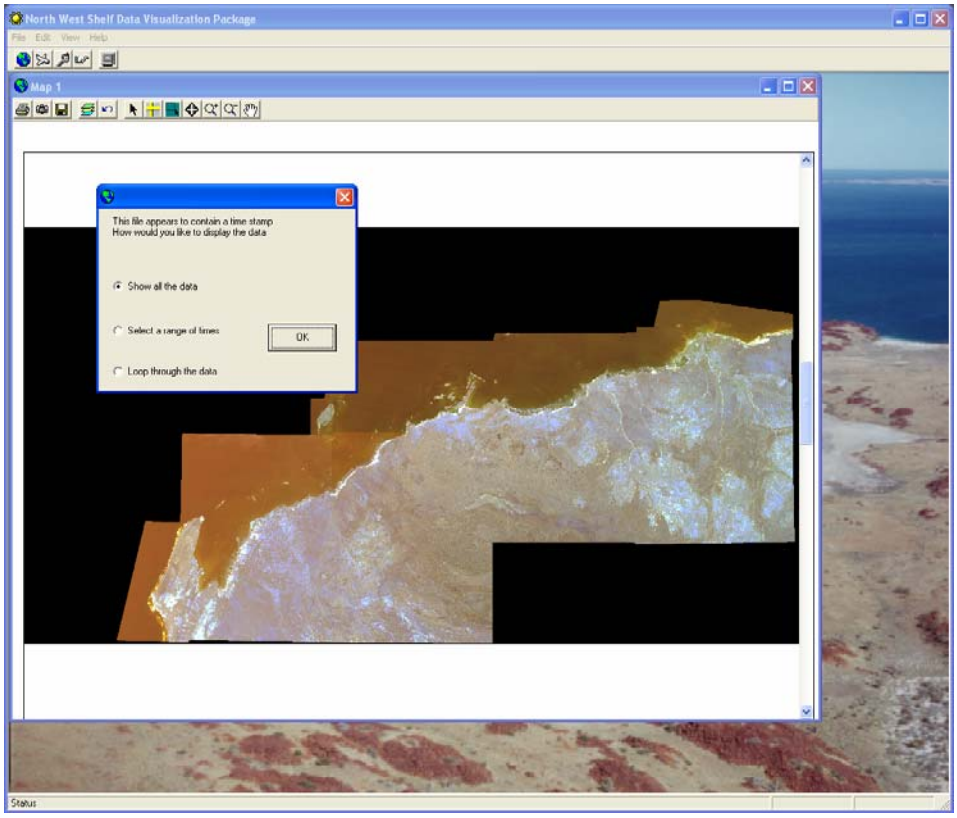


Figure 2.6.11: Add layer open file dialogue box.

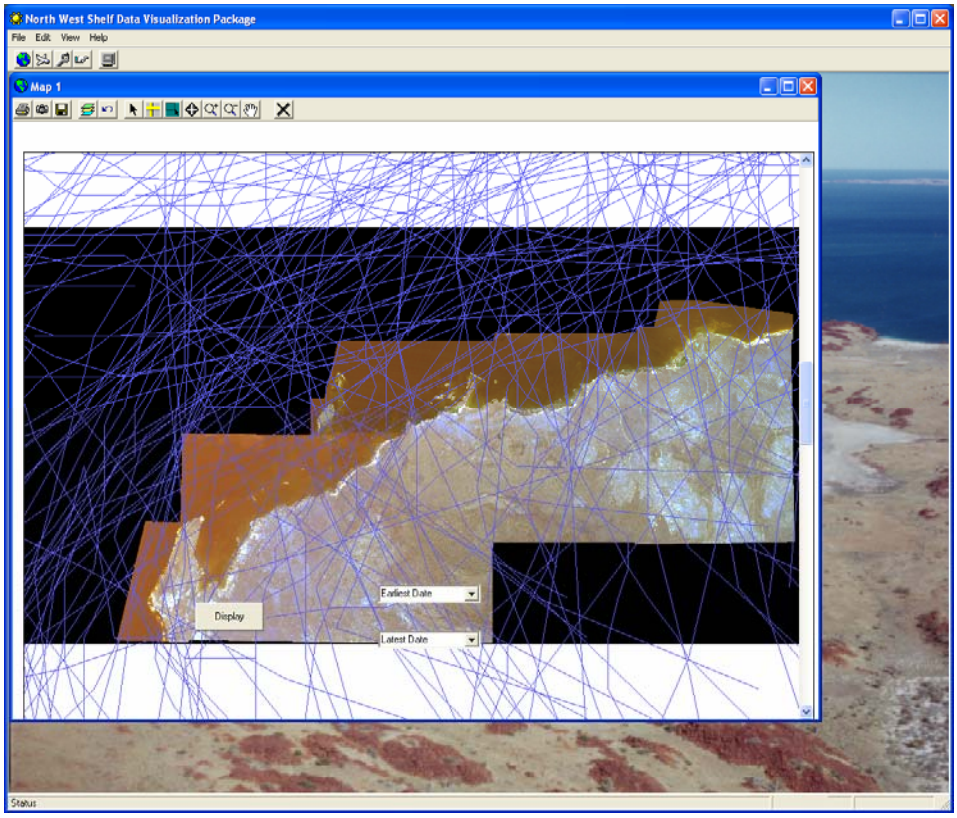


Figure 2.6.12: Added map layer (from select a range of times).

To display the maps for each of the MSE options, click on **add map layer**  . A file open dialogue box will be opened. Click on the **mse** folder, a list of folders describing what 3 by 3 by 3 parameters are used for the files contained in them will appear. For example the folder O1D3M2 means:

- Operating model: pessimistic interpretation;
- Development Scenario: continued development; and
- Management Strategy: enhanced sectoral strategies.

(See section 2.6.7 on the MSE screen for the full list of scenarios.)

Once the folder required is opened, click on the requested **file** and then click on **Open** (figure 2.6.13).

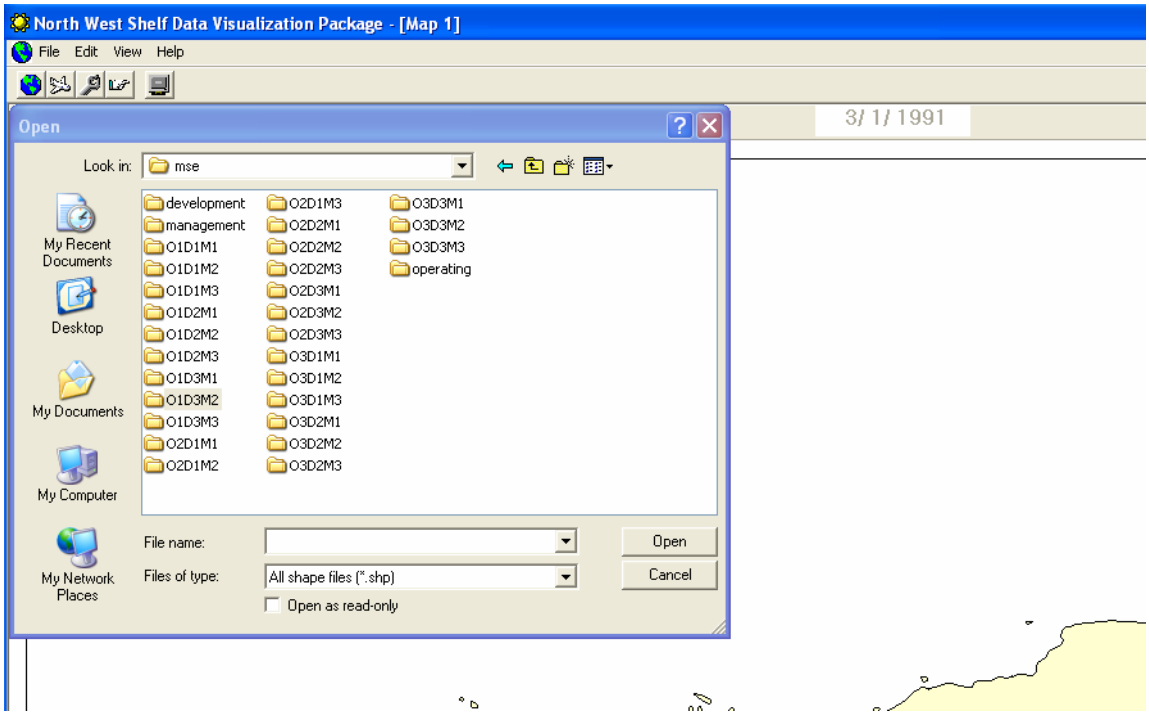


Figure 2.6.13: Adding an MSE layer.

Printing and copying

To print the map currently displayed, press **print** . A print dialog box will open. Select the printer required and press **Print** (figure 2.6.14). Alternatively the shortcut keys **Alt + p** can be used to bring up the dialog box.

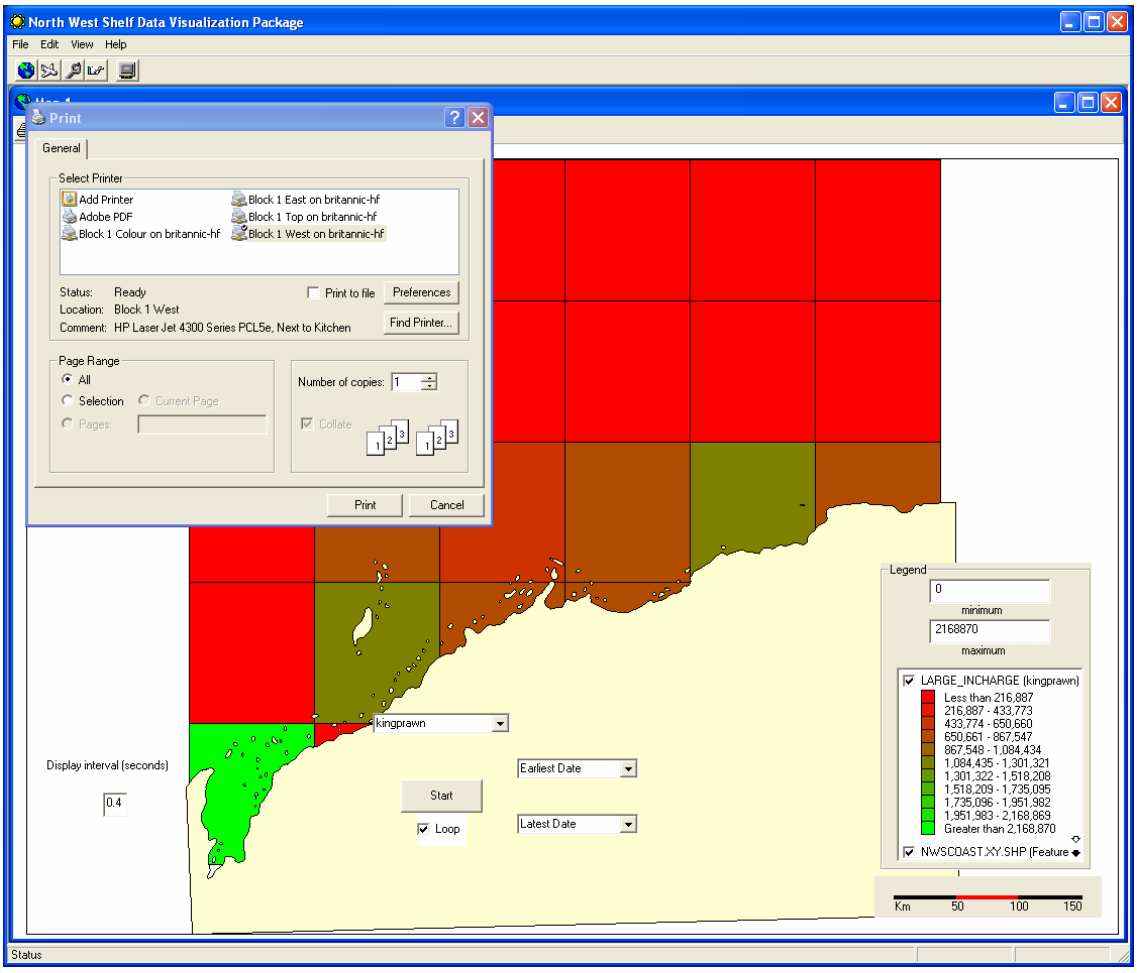


Figure 2.6.14: Printing in the map layer.

To copy the map to the clipboard; where it can be then used in presentations and reports, click on **copy map to clipboard**  . A message box will say that the image has been saved to the clipboard (figure 2.6.15).

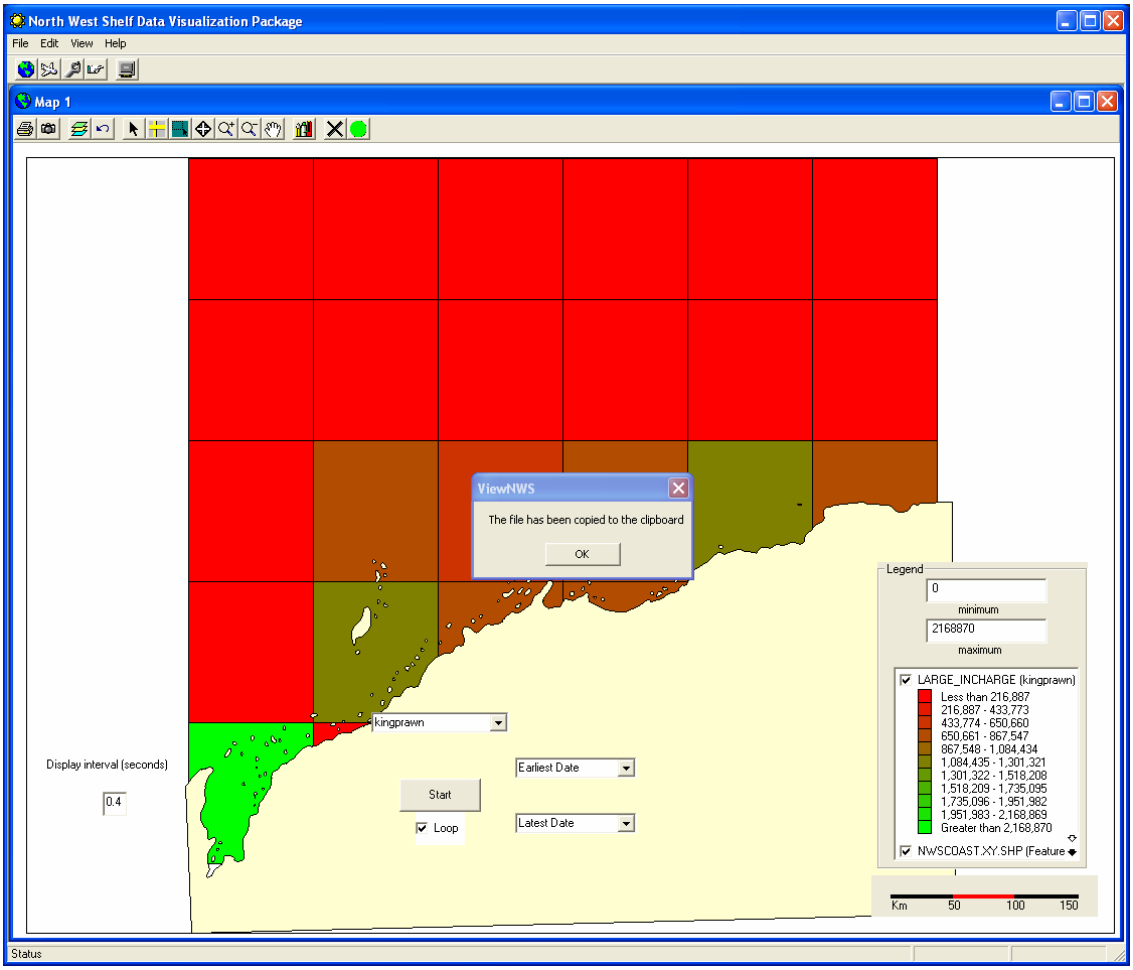


Figure 2.6.15: Copy button on the map window.

Point, measure and selecting an area

The **point button** allows the user to point on the map. This is required when using the buttons and drop down menus.

Distances on the maps can be measured by using **ruler** (figure 2.6.16). The units given will depend on the underlying map data. To measure a distance select **ruler**, click at the starting point of the line to be measured and move the mouse to the end point, then double click the mouse at the end point – this will give a message box that displays the distance. The borders of different shapes can be measured by clicking once at each change of direction and double clicking at the end point. The total distance will then be given.

To turn off the ruler tool, click on the **point button** and the line drawn will disappear.

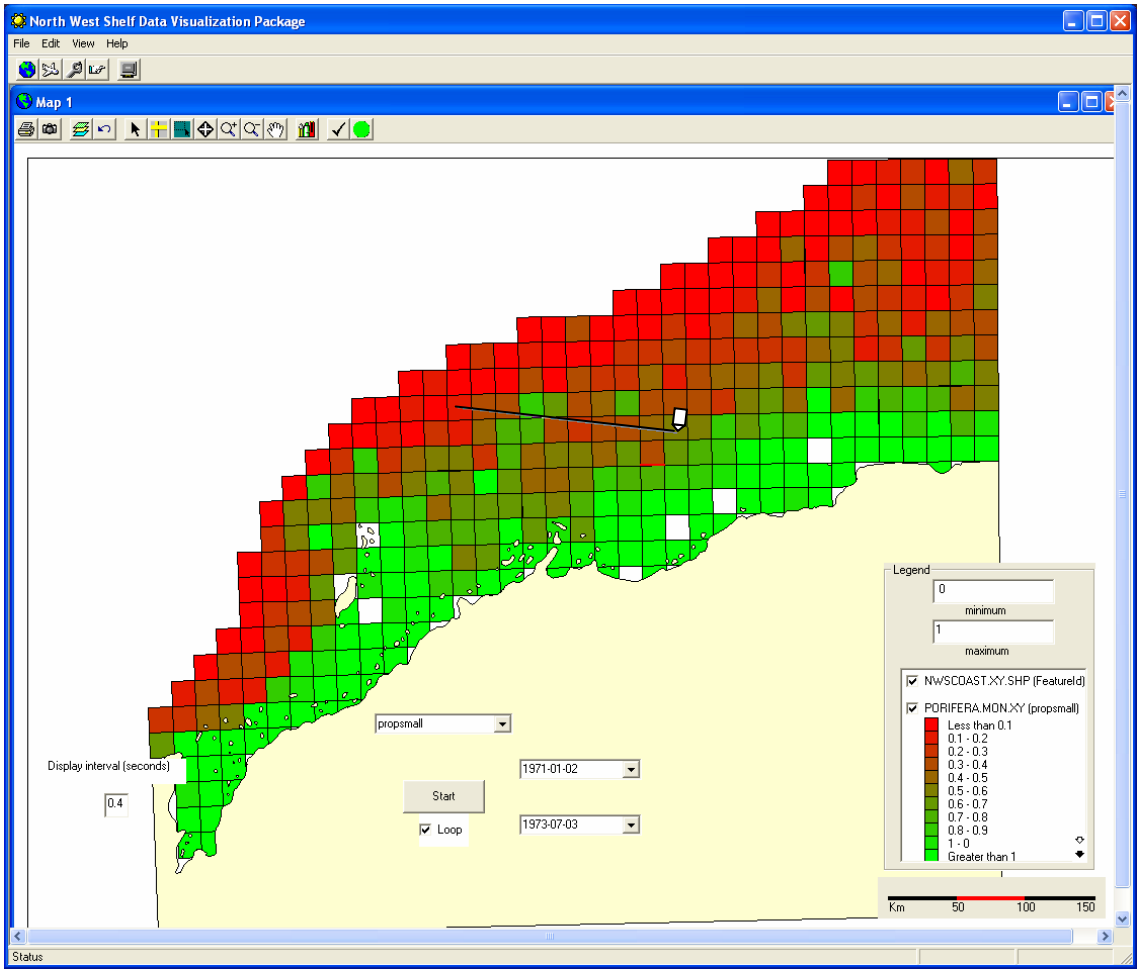


Figure 2.6.16: Ruler tool.

The **area selection** button allows the user to zoom in on a particular area of interest on the map (figure 2.6.17). To select an area, click on **area selection**, put the cursor near to the area of interest, then click and drag to make a rectangle (the cursor will change to a magnifying glass). When the mouse button is released the view refreshes to the area selected.

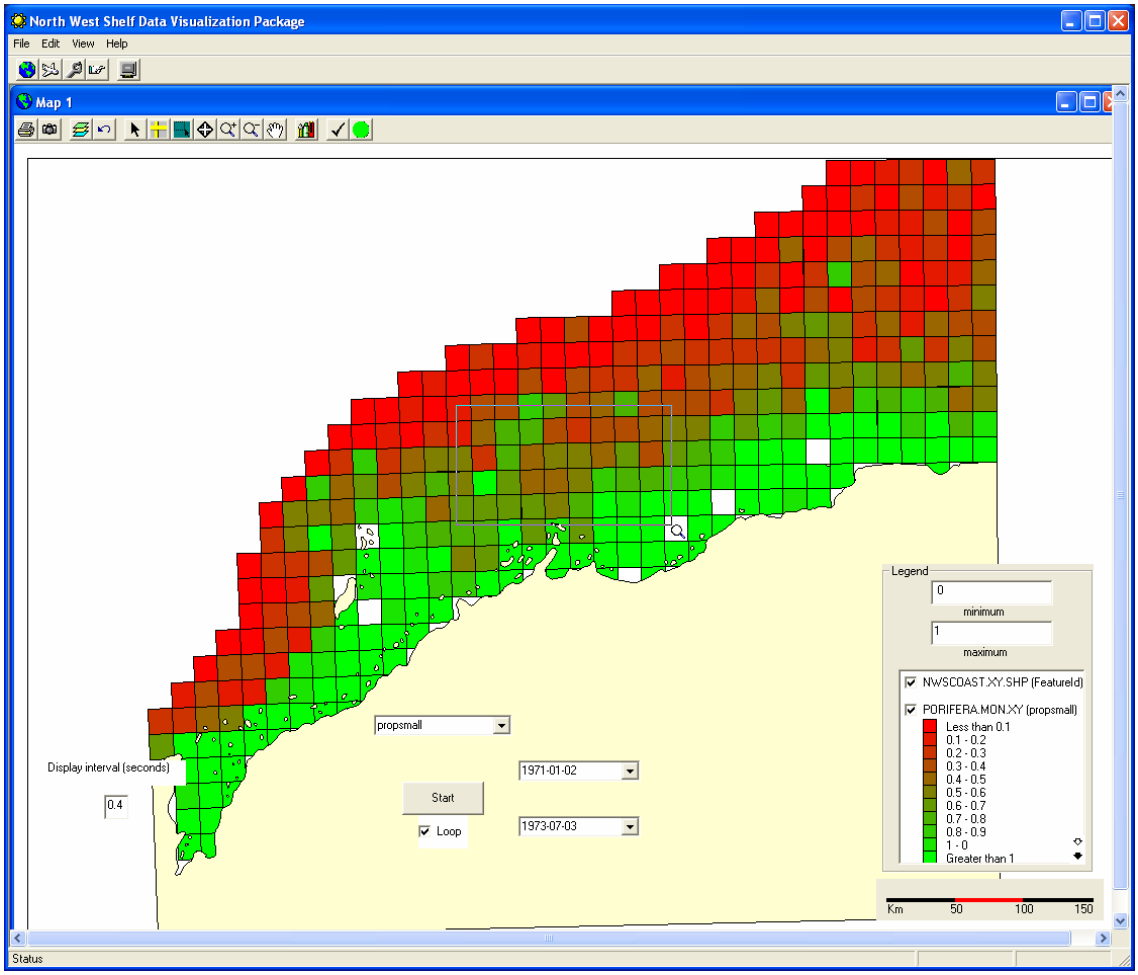


Figure 2.6.17: Selecting an area.

Zooming and panning

The map resolution can be changed by using **zoom in** and **zoom out**. To return to the full extent of the map, click on **restore to normal size**. To zoom in/out click on either the zoom in or zoom out button and then click on the area of interest to be zoomed in/out on.

Pan allows the user to shift the view at the current scale. To do this, select **pan** and place the cursor in the middle of the view. Click and drag to the new area (figure 2.6.18).

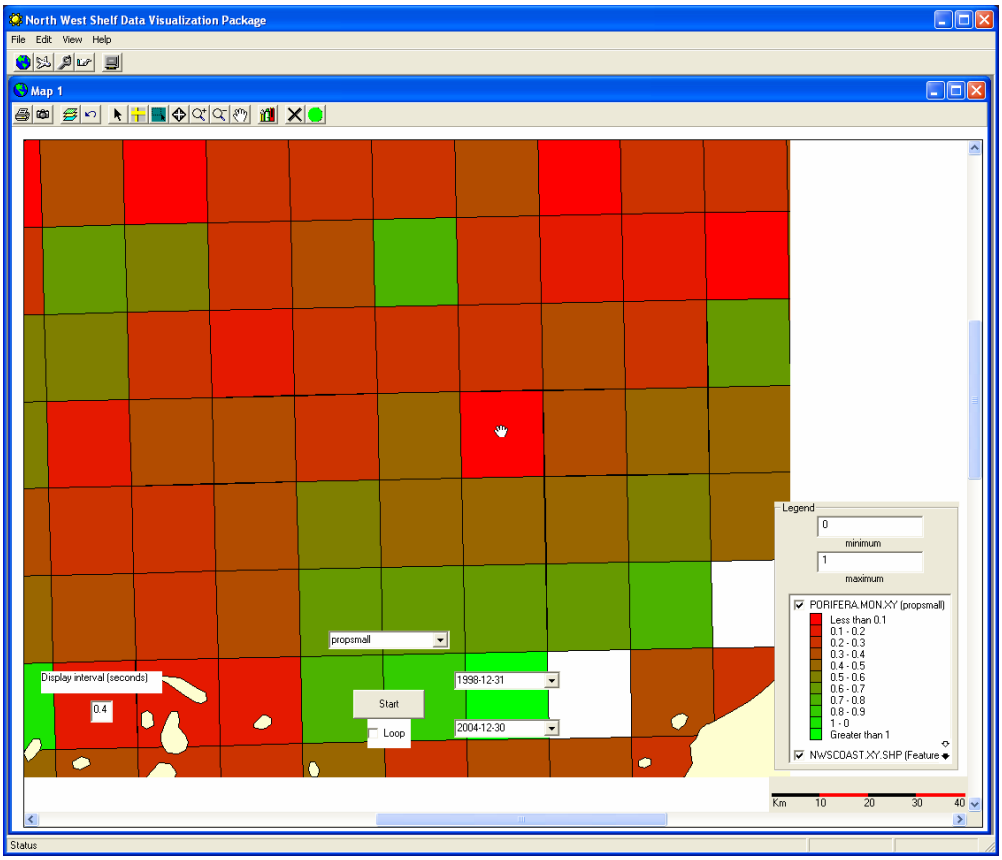


Figure 2.6.18: Panning.

Editing the map legend

To edit the map legend of the layer that is the active top layer, click on **edit map legend**  . A dialog box will come up with six tabs that allow the user to change the map legend (figure 2.6.19).

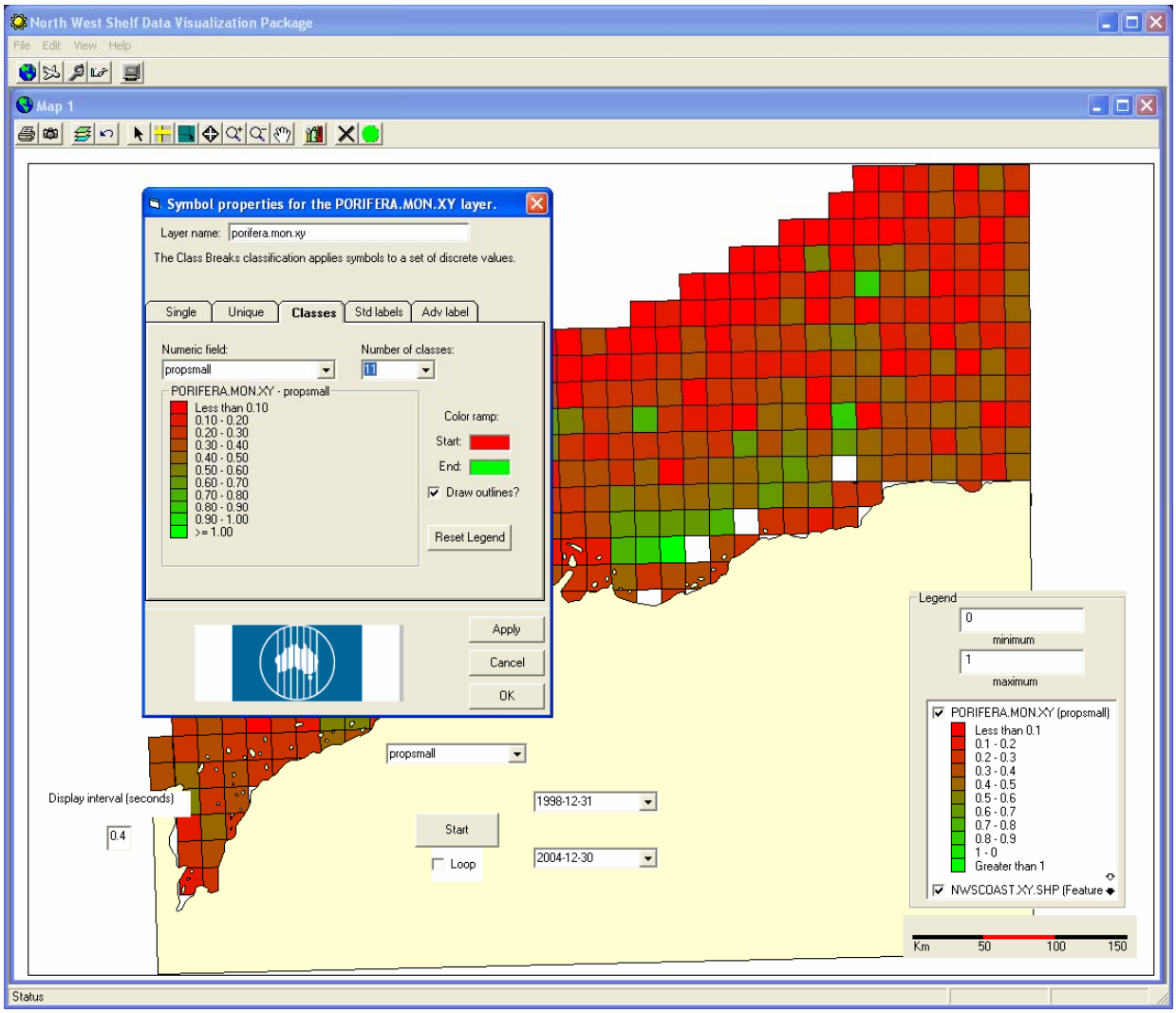


Figure 2.6.19: Editing the map legend dialog box.

At the bottom right hand corner of each tab the user can click **Apply** to apply their changes and move to another tab, or click **OK** and return to the map.

The first tab is **Single**. This displays all the features in a layer with the same symbol (figure 2.6.20).

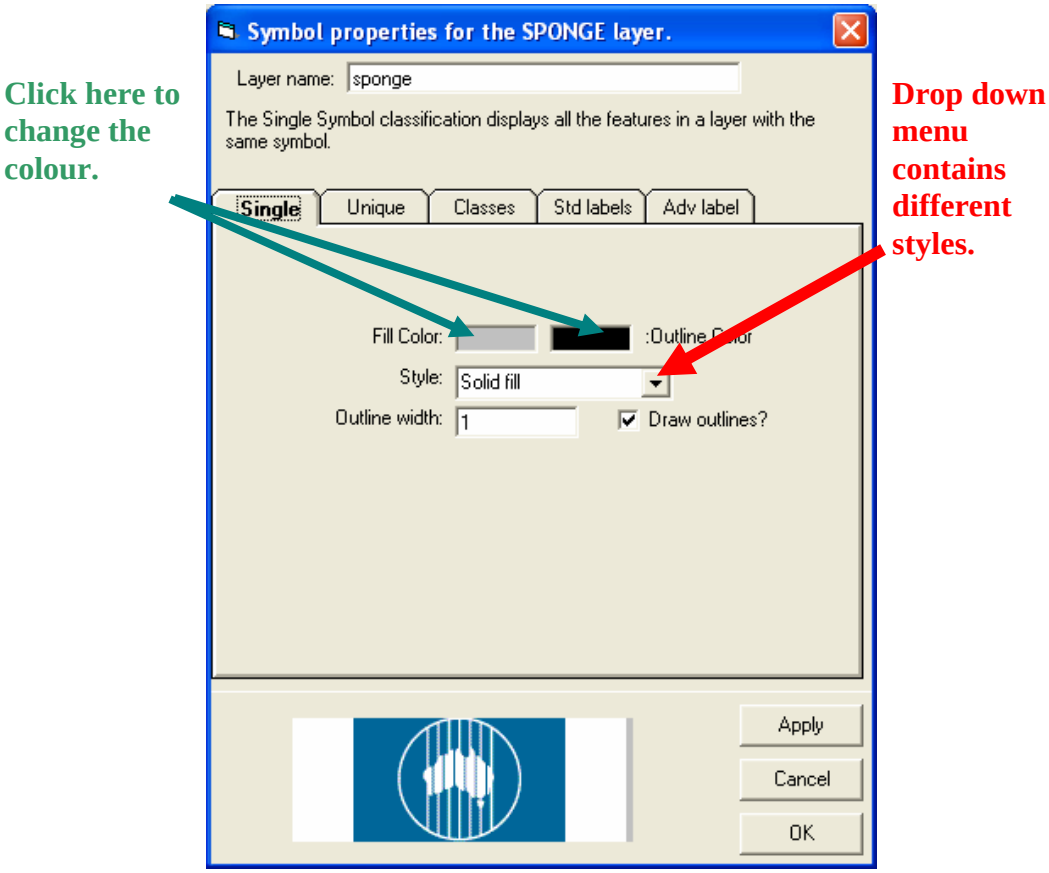


Figure 2.6.20: Single tab.

Both the **Fill** and **Outline Colours** can be changed by clicking in the colour rectangle. A colour choosing dialog box will then open and a colour can then be picked.

The **Style** of the symbol classification can also be changed. Click on the drop down menu to access the following styles:

- solid fill
- transparent fill
- horizontal fill
- vertical fill
- upward diagonal
- downward diagonal
- cross fill
- diagonal cross fill
- light grey fill
- grey fill
- dark grey fill

The **outline width** of the classification can be made thicker by typing in a higher number. The outline of the classifications will be drawn when the **Draw Outline?** box is selected.

The second tab is the **Unique** tab (figure 2.6.21). This tab allows the user to apply a symbol to each unique value for a specified field.

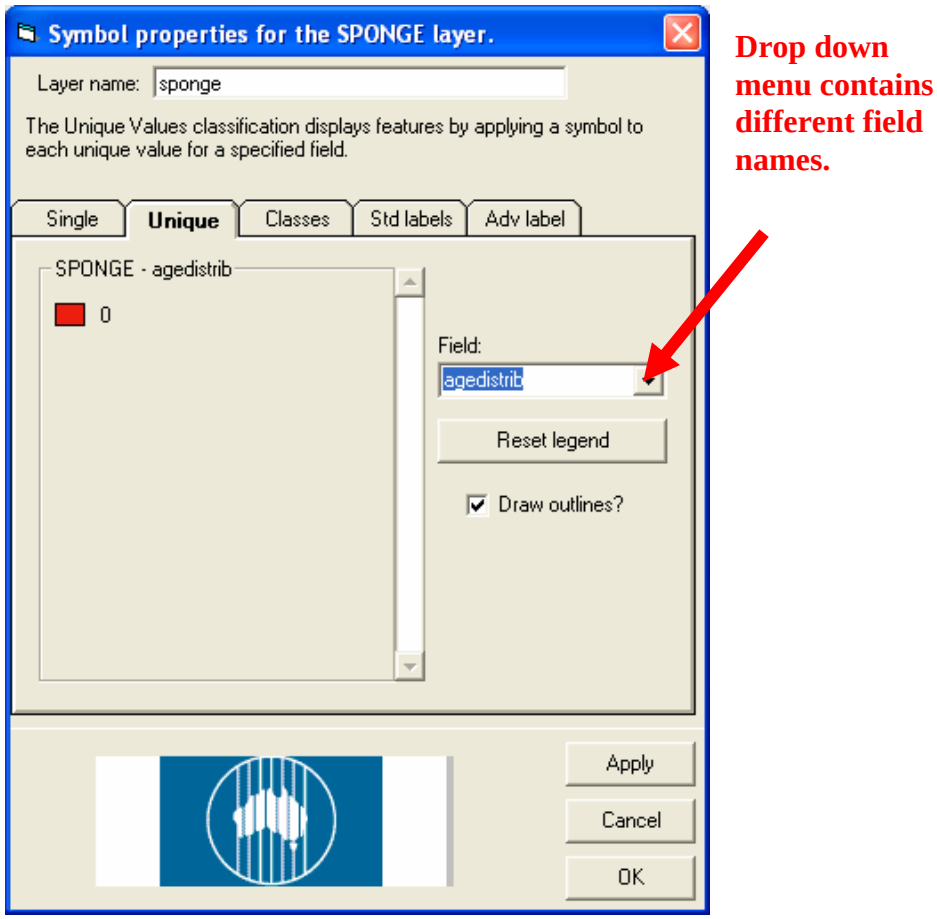


Figure 2.6.21: Unique tab.

The **Field** names are taken from the shape file that is currently active on screen. Select the required field to display from the drop down menu.

The **Reset legend** button allows the user to update the legend being displayed on the tab after the **Field** name is selected. By pressing it without changing the **Field** name the user can change the colours being used for the legend before applying it to the map.

To draw outlines around each of the classes on the map select the **Draw outlines?** box.

The third tab is the **Classes** tab (figure 2.6.22). This tab allows the user to change the colour and number of classes of the legend.

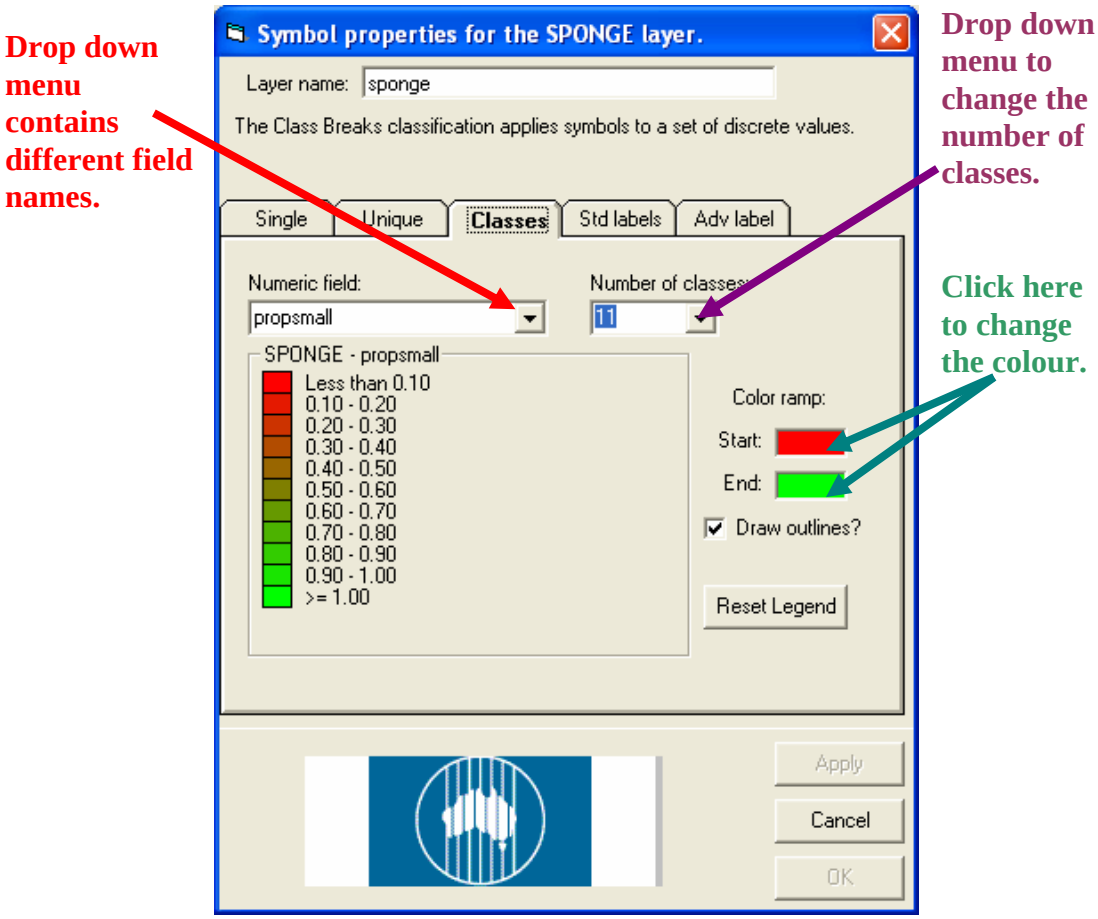


Figure 2.6.22: Classes tab.

The **Numeric field** contains a drop down menu of fields that is sourced from the active shape file.

The **Number of classes** allows the user to change the scale of the legend.

The **Colour ramp** is changed by clicking in the colour box. A colour palette dialog box will appear allowing colour choice.

Selecting the **Draw outline?** check box gives a border around each of the classes on the map.

Click on **Reset Legend** to preview the new legend before applying it to the map.

The fourth tab is **Std labels** (figure 2.6.23). On this tab the user can select how the data labels look.

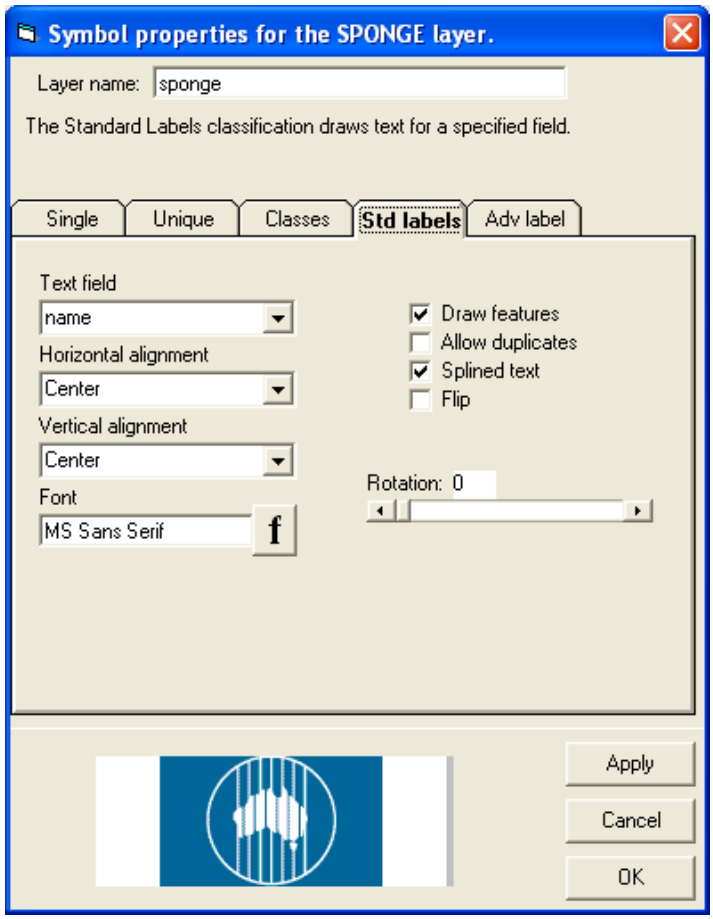


Figure 2.6.23: Std labels tab.

The **Horizontal alignment** can be changed to left, centre or right.

The **Vertical alignment** can be changed to top, centre or bottom.

To change the font, click on **Font** f. A font dialog box will then appear which will allow the user to choose the font, its style, the size, any effects and the colour.

The **Rotation** slide bar allows the user to rotate the legend text up to 359 degrees.

The fifth tab is the **Adv label** tab (figure 2.6.24). This tab helps the user to format the map legend so that it is clear and easy to read.

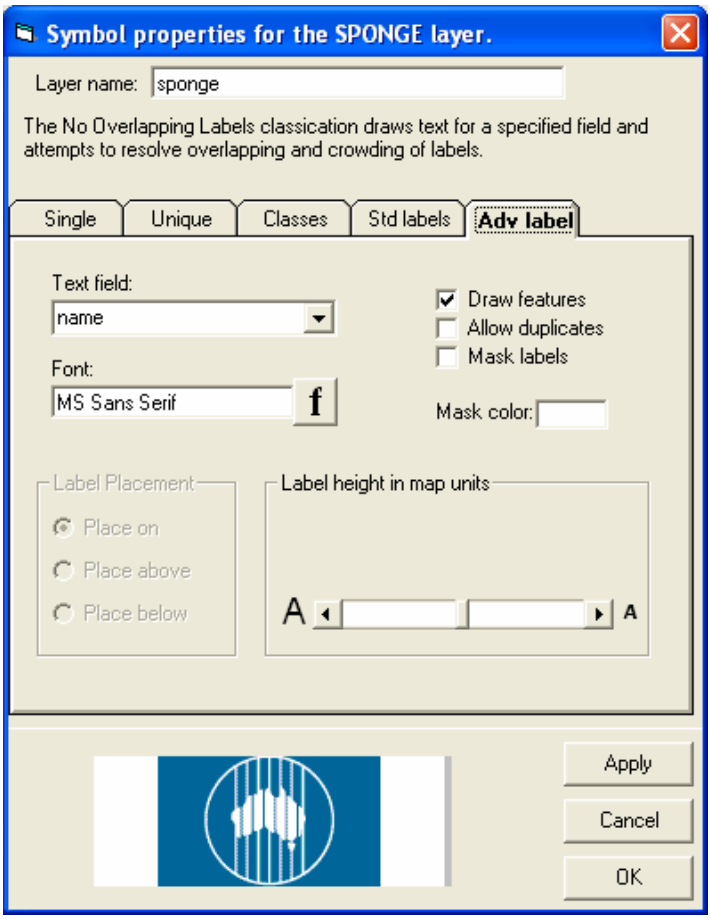


Figure 2.6.24: Adv label tab.

Again the text fields come from the shape file being displayed.

To change the font, click on **Font** f. A font dialog box will then appear which will allow the user to choose the font, its style, the size, any effects and the colour.

Screen and display controls

To hide the on screen controls click on **hide on screen controls** . To restore the screen controls click on **show on screen controls** .

To start the display on the map, click on **start** . To stop the display, click on **stop** (figure 2.6.25).

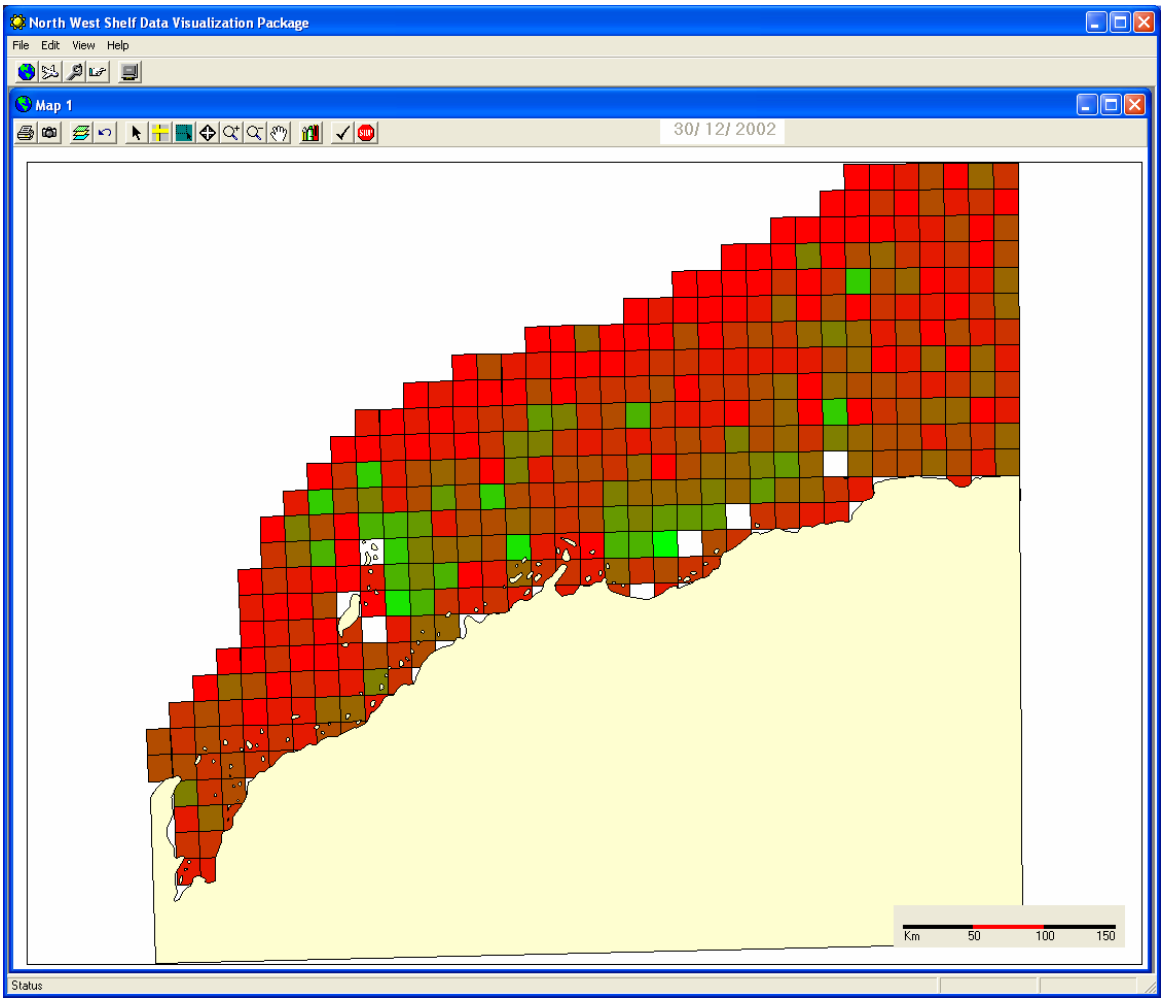


Figure 2.6.25: Screen controls hidden, screen display running.

The map layer itself also has screen controls (figure 2.6.26).

The data being displayed from the shape file can also be changed by clicking the drop down menu. The view and legend will refresh to display the data selected.

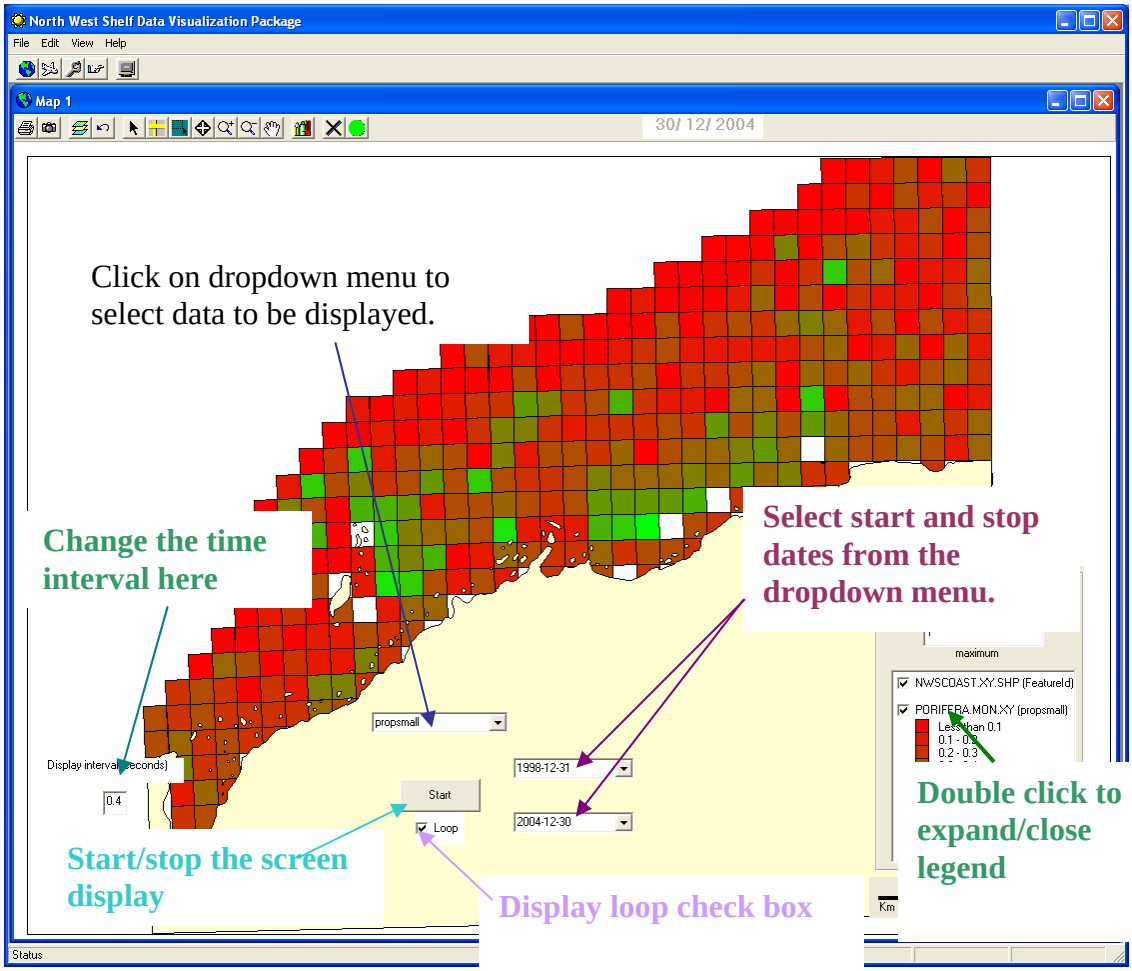


Figure 2.6.26: Controls on layer.

If the data being displayed has a time stamp to it, the **start** and **end** dates for the display can also be selected from the **Earliest date** and **Latest date** drop down menus. The display interval can also be changed – to do this – click in the **Display interval (seconds)** box and type how many seconds are required before the display changes.

To start the display, click on the **Start** button. To stop the display, click on the **Stop** button. The display can be left running in a loop if the **Loop** check box has been selected.

By clicking on specific points on the map, time series or depth series graphs of the relevant data type can be displayed. To close the graph click on the **close button** situated on the top right hand corner of the graph window

At the bottom right of the screen there is a window that behaves like the table of contents in an *ArcMap* interface. This window is not resizable.

To change the scale of the data being displayed, click and enter in appropriate values in the **minimum** and **maximum** boxes.

To hide layers clear the check box next to the layers. To show/make layers active select the check box beside the layers required.

To display/hide the layer's legend, double click on the layer name (figure 2.6.27).

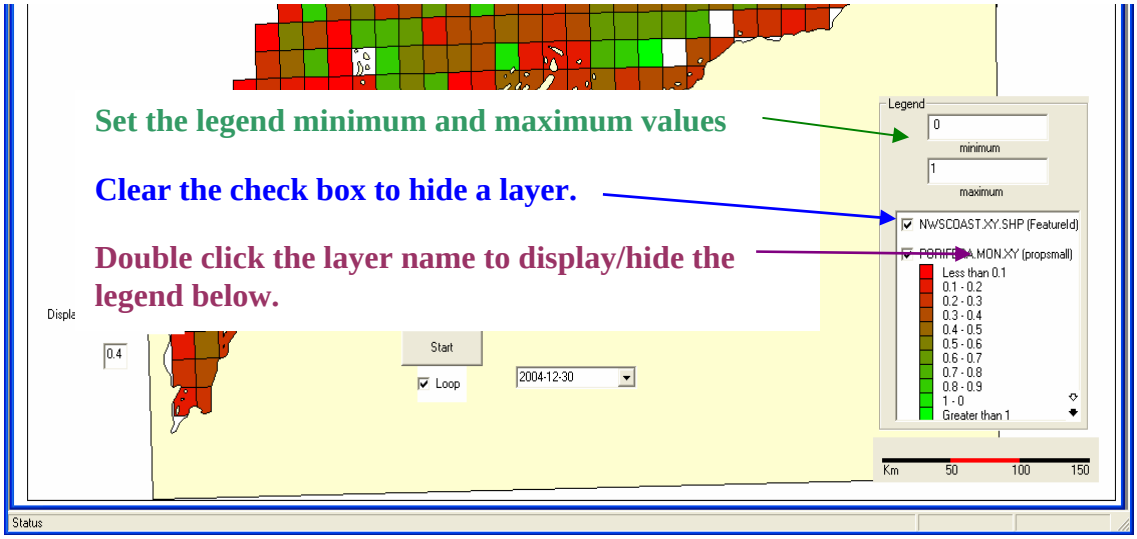


Figure 2.6.27: Layer controls.

If there are many layers, or layers with large legends displayed, a scroll bar appears on the right hand side of the window. Scroll up or down to find the layer required.

In order to change the layer's legend, the layer *must be active* at the top of the list. To move a layer, click on its name and drag to the required location (figure 2.6.28).

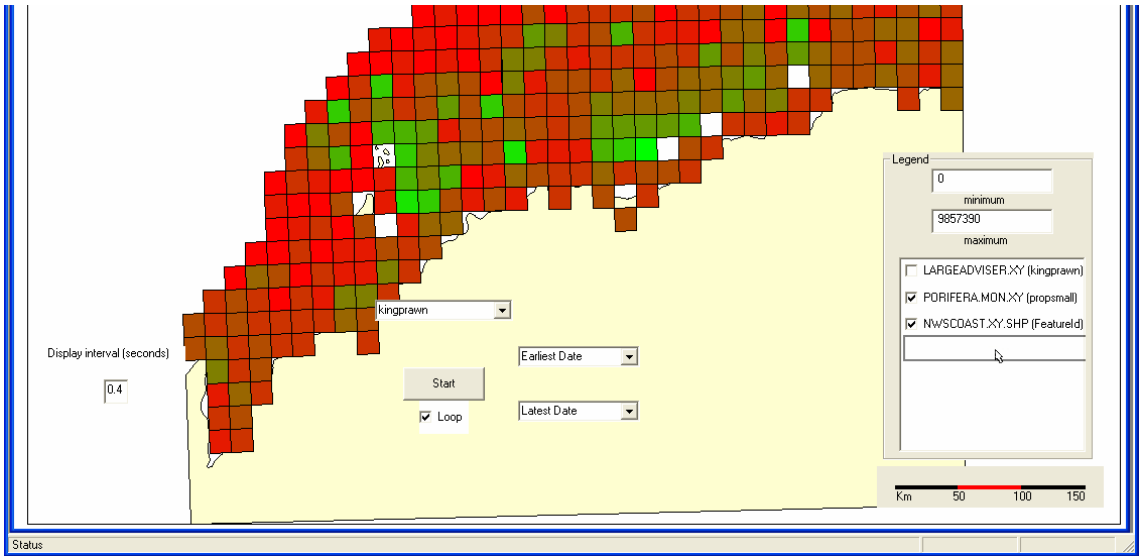


Figure 2.6.28: Moving a layer.

Many layers can be active on the screen, however only the base map and the *active* top layer are displayed.

2.6.7 The scenario chooser – MSE

Click on **scenario chooser**  (figure 2.6.29) to launch the Management Strategy Evaluation (MSE) screen (figure 2.6.30).

Figure 2.6.29: Launching the scenario chooser.

The MSE screen allows the user to select from a number of different scenarios (figure 2.6.30). The choices are constrained within a 3 by 3 by 3 matrix of possibilities.

The first dimension of the matrix is the Operating Model. This contains the following choices:

1. Pessimistic Interpretation
2. Base Model
3. Optimistic Interpretation

The second dimension is the Development Scenario, which contains the following choices:

1. No Pulse
2. Single Pulse
3. Double Pulse

The third and final dimension is the Management Strategy. This contains the following choices:

1. Status Quo
2. Enhanced Sectoral Strategies
3. Regionally Coordinated Sectoral Strategies

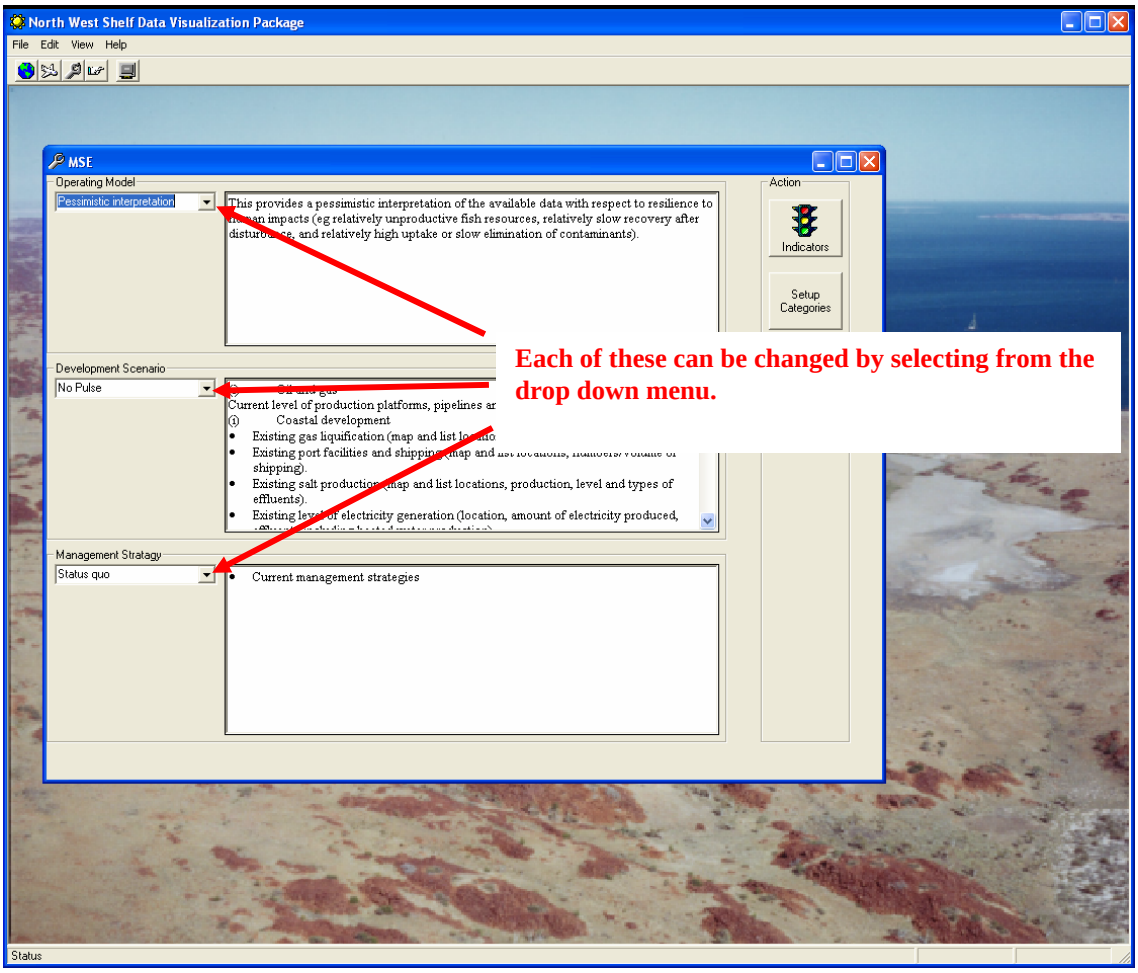


Figure 2.6.30: The management strategy evaluation (MSE) screen.

Once the user has chosen the management strategy required, press **Indicators**  . A screen opens that allows the user to select the indicator results from several scenarios (figure 2.6.31).

The other option is to press **Setup Categories**  . This enables the user to group indicators under categories which are explained further in the parameters section.

2.6.8 Indicator screen

Once the scenario has been chosen the appropriate indicators screen will be launched. The Indicator screen displays the environmental indicators that have been agreed to for the project. In order to highlight a specific problem area, the indicators are flagged using a “traffic light” system. This system grades the environmental indicators against specific targets.

- **Green** indicates the indicator is within the target;
- **Yellow** signifies that the indicator should be examined more closely; and
- **Red** means that the target indicator has not been reached.

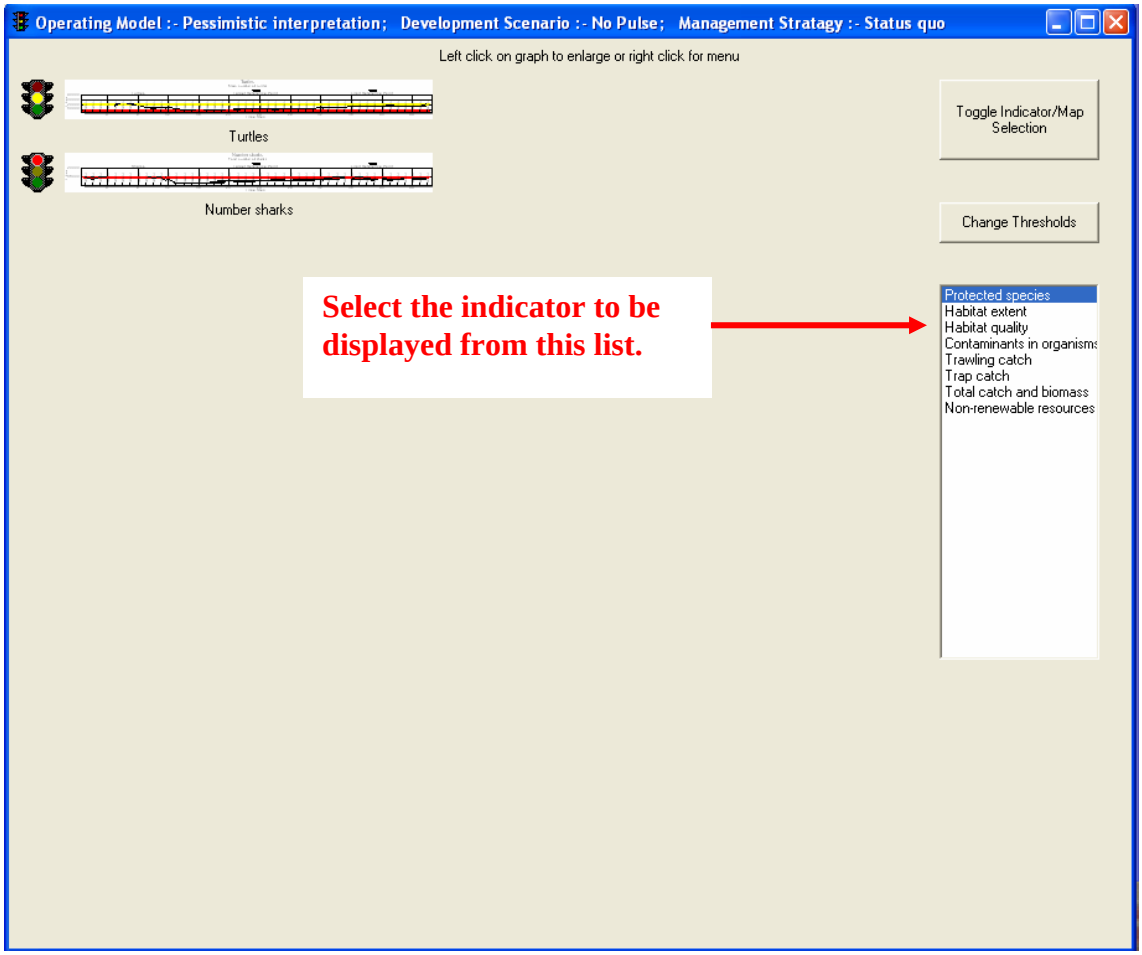
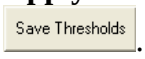


Figure 2.6.31: The indicator screen.

To change the thresholds of the indicators click on **Change Thresholds** 

. This will allow the user to change the thresholds of the indicator by typing in values into the corresponding boxes (figure 2.6.31). The target reference point is the yellow box and the limit reference point is the red box. To apply the new thresholds, click on **Apply Thresholds** 

. To save the selection click on **Save Thresholds** 

.

To return to the indicator selection screen press **Show Categories** 

.

To display a different indicator click on one of the indicators displayed on the right hand side of the screen.

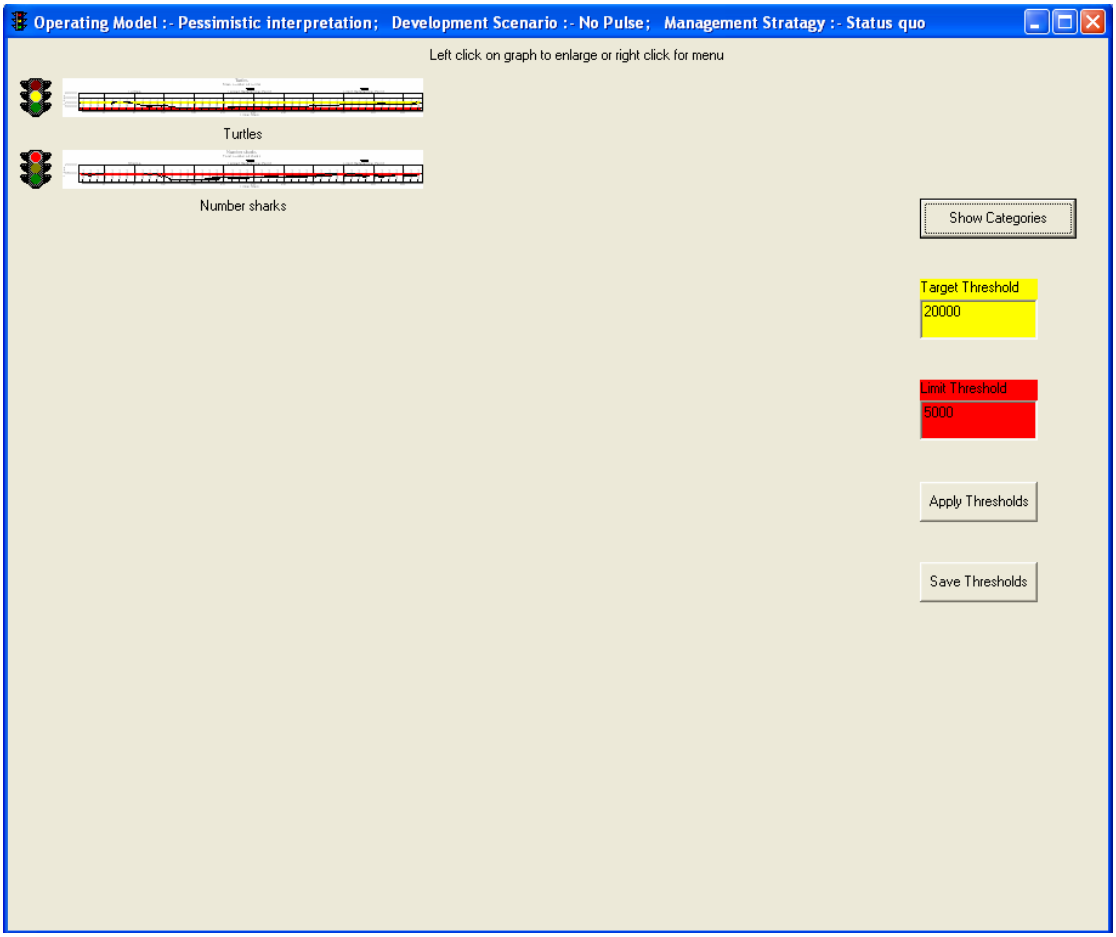


Figure 2.6.32: Changing the indicator threshold screen.

Toggle Indicator/Map Selection  allows the user to display the indicators as either a graph or a map of the spatial distribution of the indicator.

To display data as a full screen graph, click **Toggle Indicator/Map Selection**  until a traffic light symbol  appears next to the graphs, then left click on the selected graph (figure 2.6.32).

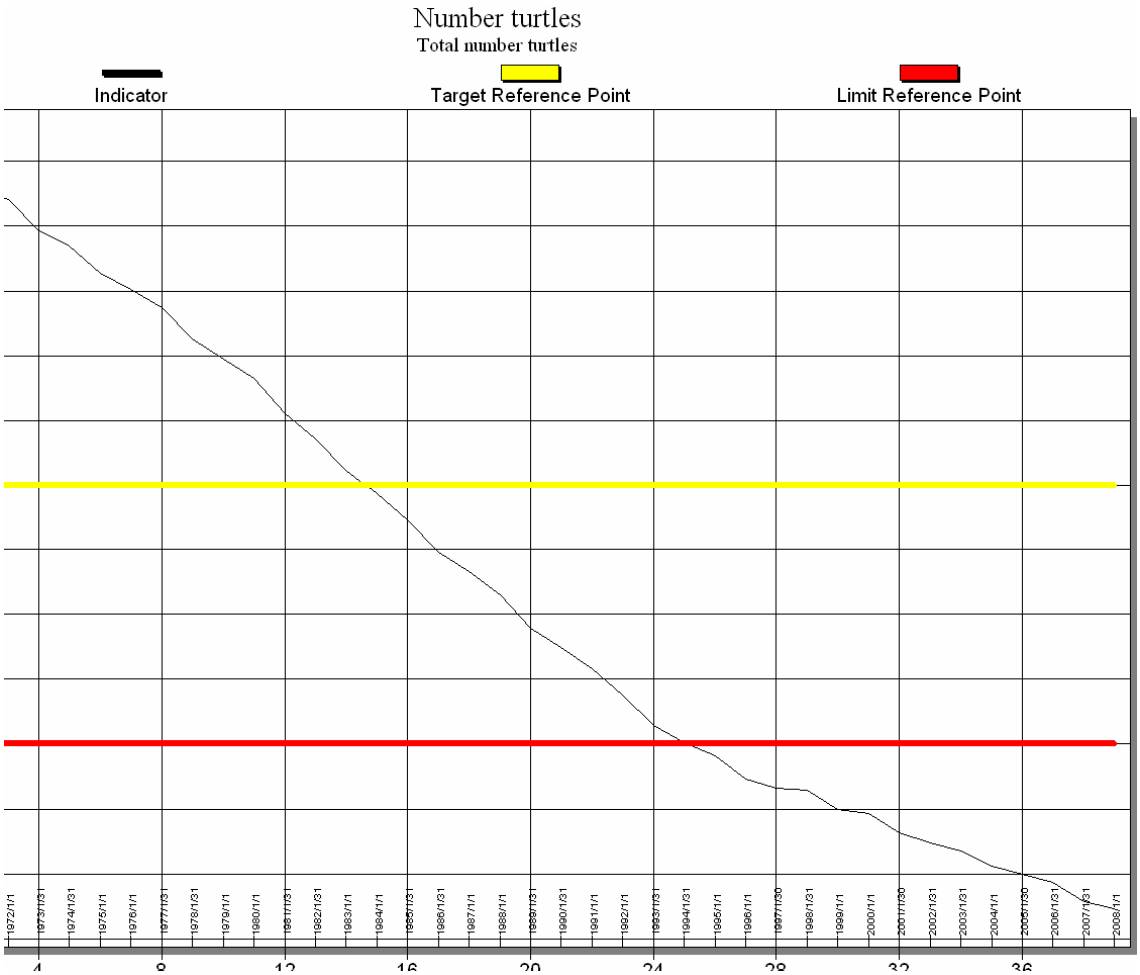
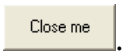


Figure 2.6.33: Full screen graphical representation of the data.

To display data as a graphical representation, click on **Toggle Indicator/Map Selection**  until a small globe appears beside each of the graphs, and then click on a **globe**  (figure 2.6.33). A separate window displaying the spatial representation will appear (figure 2.6.34).

To close the window click **Close me**  . A left click on the selected graph will again produce a full screen graph of the data. The data being displayed may be changed by selecting a timestamp from the drop down menu.

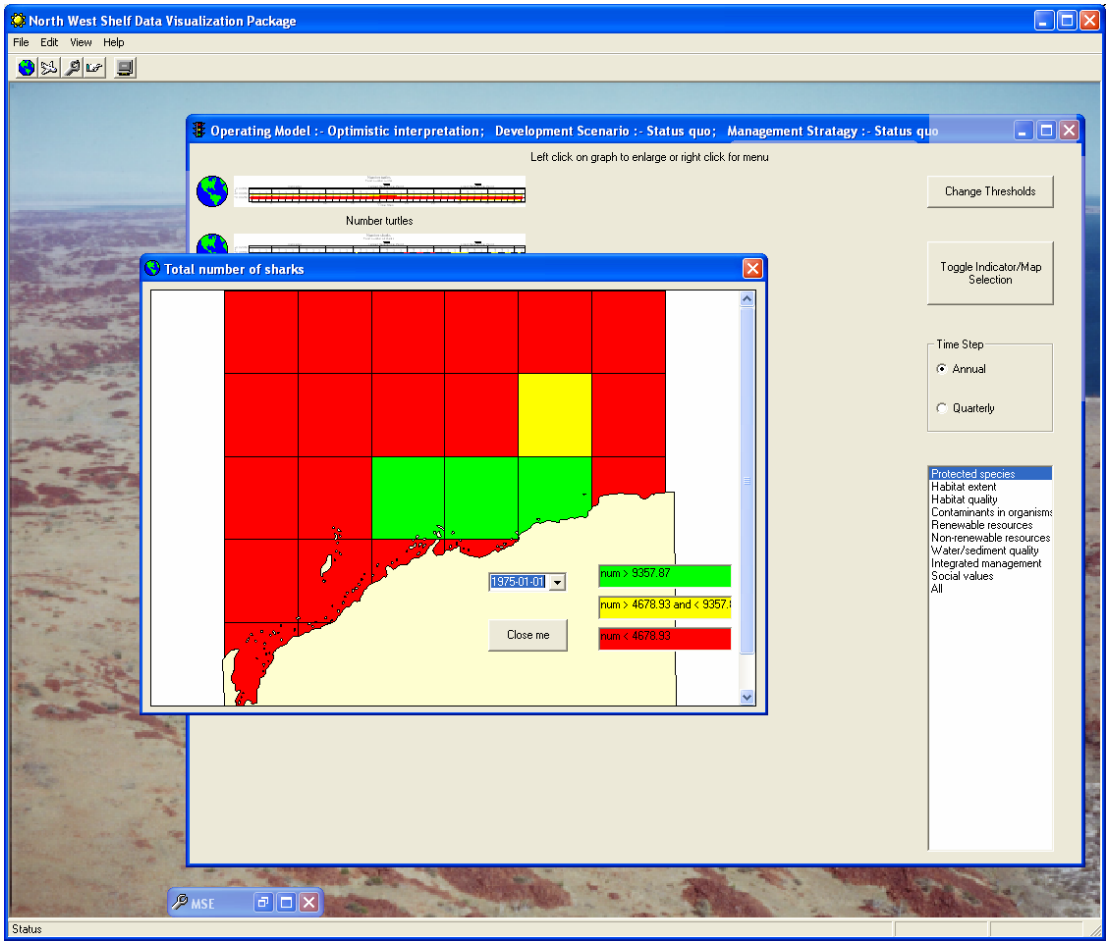


Figure 2.6.34 Spatial representation of the data.

To display how the indicator performs across all scenarios click on **Toggle Indicator/Map Selection**  until the traffic lights appear besides each of the graphs, and then click on **Indicators**  and a separate window will open showing how any given indicator performs across the three by three matrix of scenarios (figure 2.6.35).

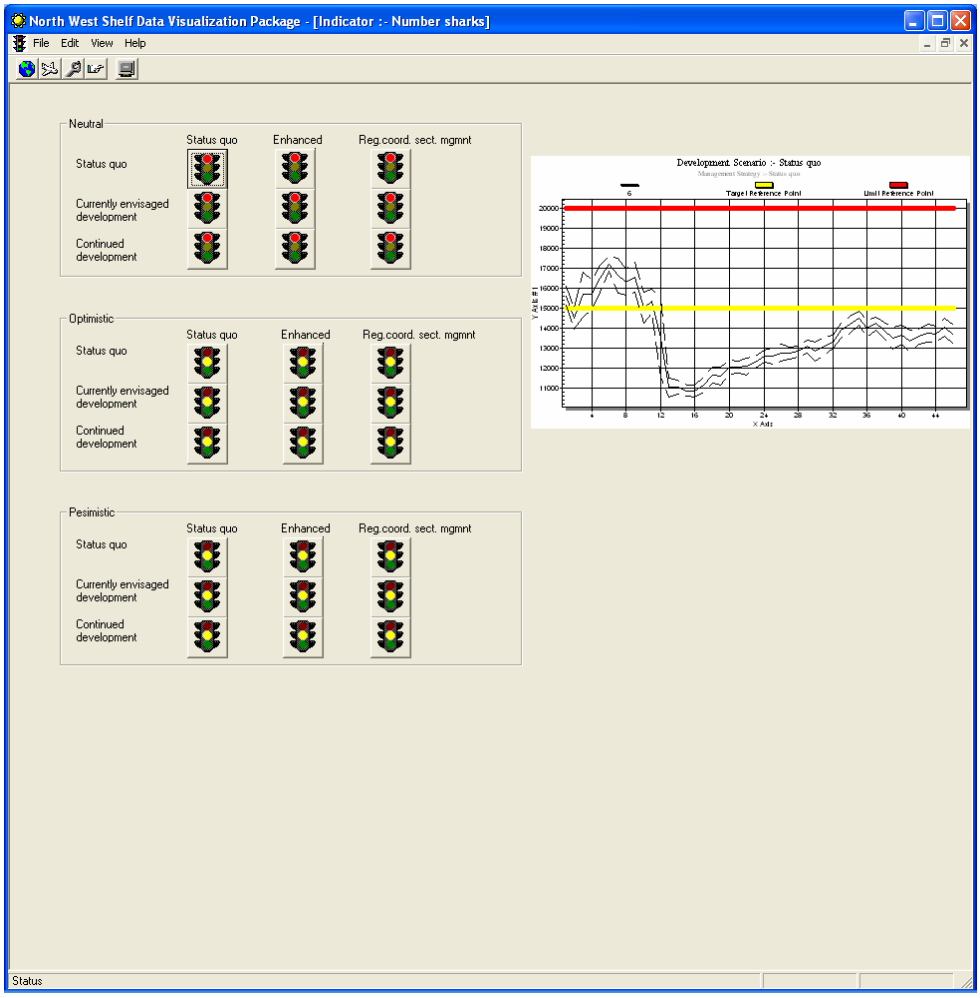


Figure 2.6.35: The indicators screen showing all scenarios.

To see the graphical representation of the data click on one of the traffic light buttons and the graph will refresh itself.

2.6.9 Parameters

The initial set up parameters can be created in the **Setup** screen. To access the **Setup** screen click **View**, **Setup** (figure 2.6.36).

Figure 2.6.36: Accessing the setup screen.

The **Setup** screen will be presented (figure 2.6.37).

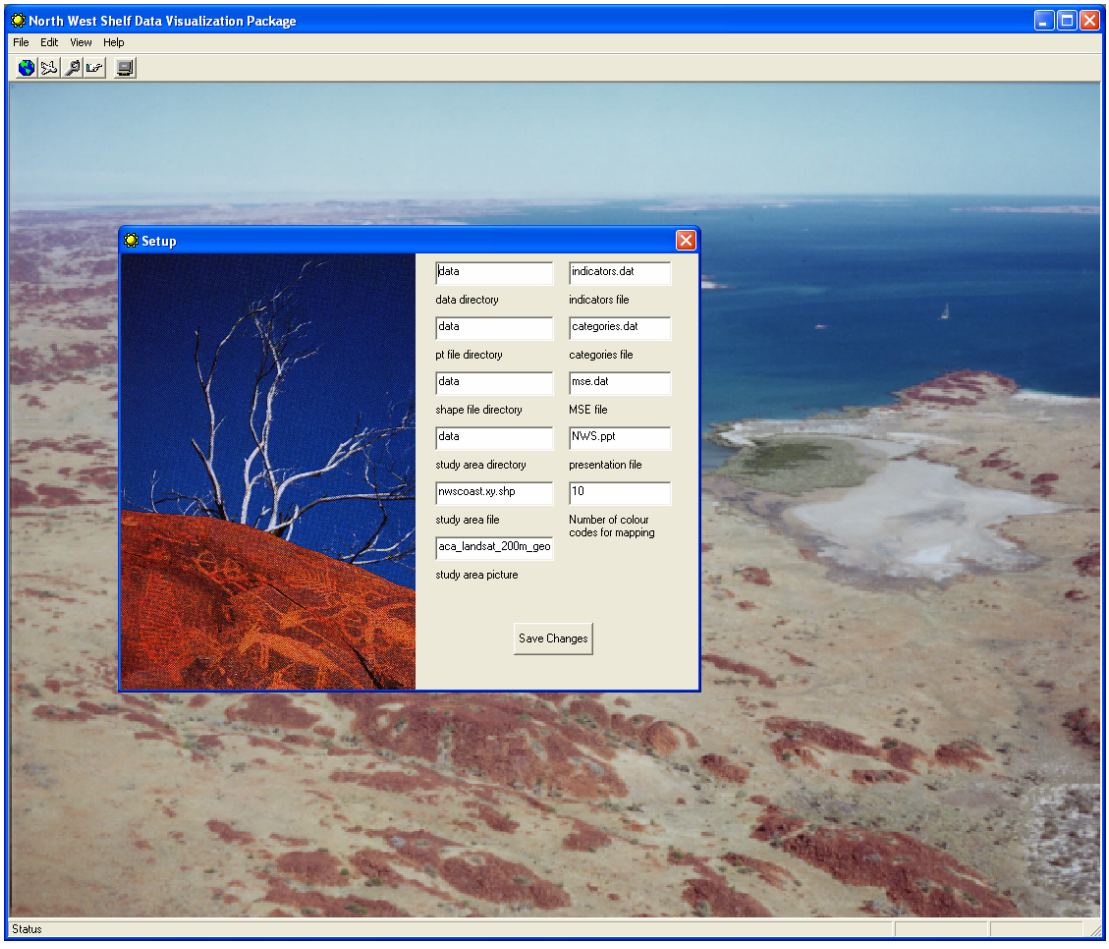


Figure 2.6.37: The Setup screen.

This screen allows the user to determine the shape file (**study area file**) and an image file (**study area picture**) for the study area.

Other files that can be defined in this screen are:

- the management strategy evaluation file (**MSE file**);
- indicators (indicators file);
- the *PowerPoint* file (**presentation file**);
- the number of colour codes for mapping;
- categories (categories file);

as well as the directories pointing to where data is stored (**data directory, pt file directory, study area directory, shape file directory**) can be defined in this screen. The path given in this screen is appended to the path of the application. To choose a directory that is not a subdirectory of the application the full path name must be included, for example, “c:\mydatadirectory”.

To set up indicators and categories of indicators click **Setup Categories** 

on the MSE screen (see figure 2.6.38). On the **Setup** screen that opens the following can be chosen:

- add and remove categories;
- add and edit indicators;
- update or set the low and high thresholds for the indicators; and
- define the indicator name displayed.

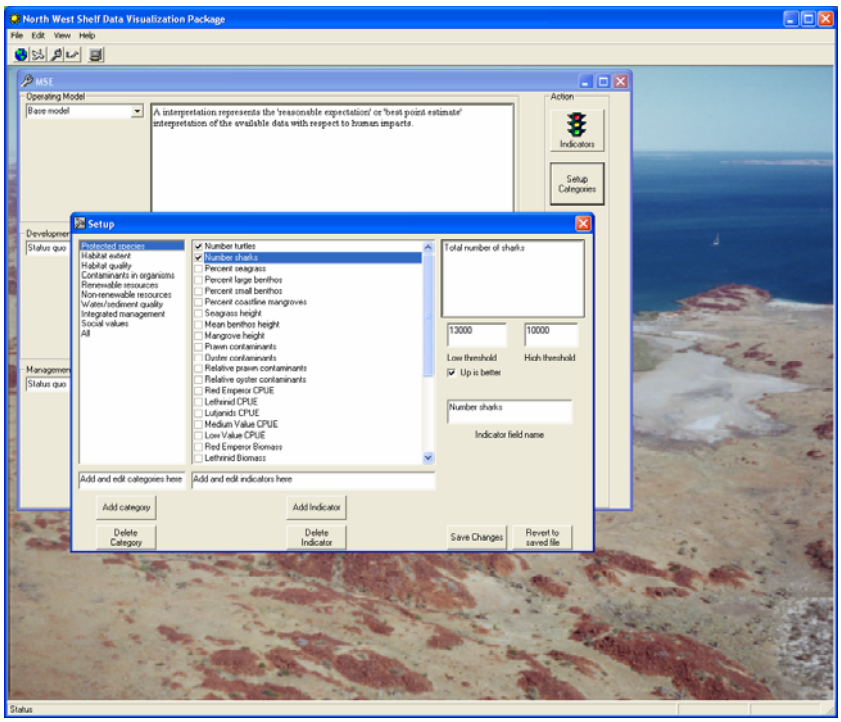


Figure 2.6.38: MSE Setup screen.

NOTE: Once **Save Changes** has been clicked, the user CANNOT **revert to saved file**.

2.6.10 File conversion

ViewNWS allows the conversion of model output files to shape files. Click on **convert file**  to launch this application (figure 2.6.39).

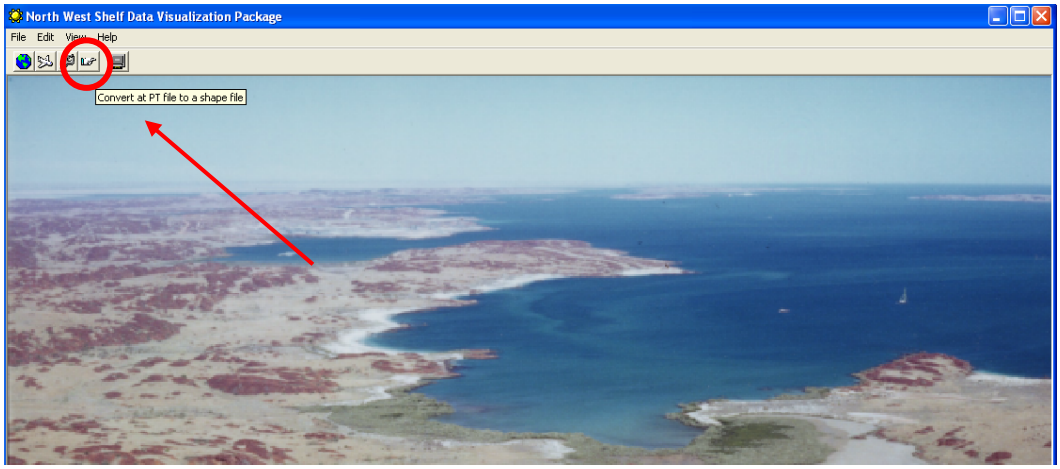


Figure 2.6.39: Launching the convert file application.

Click **choose .pt/.tbl** file button (figure 2.6.40).

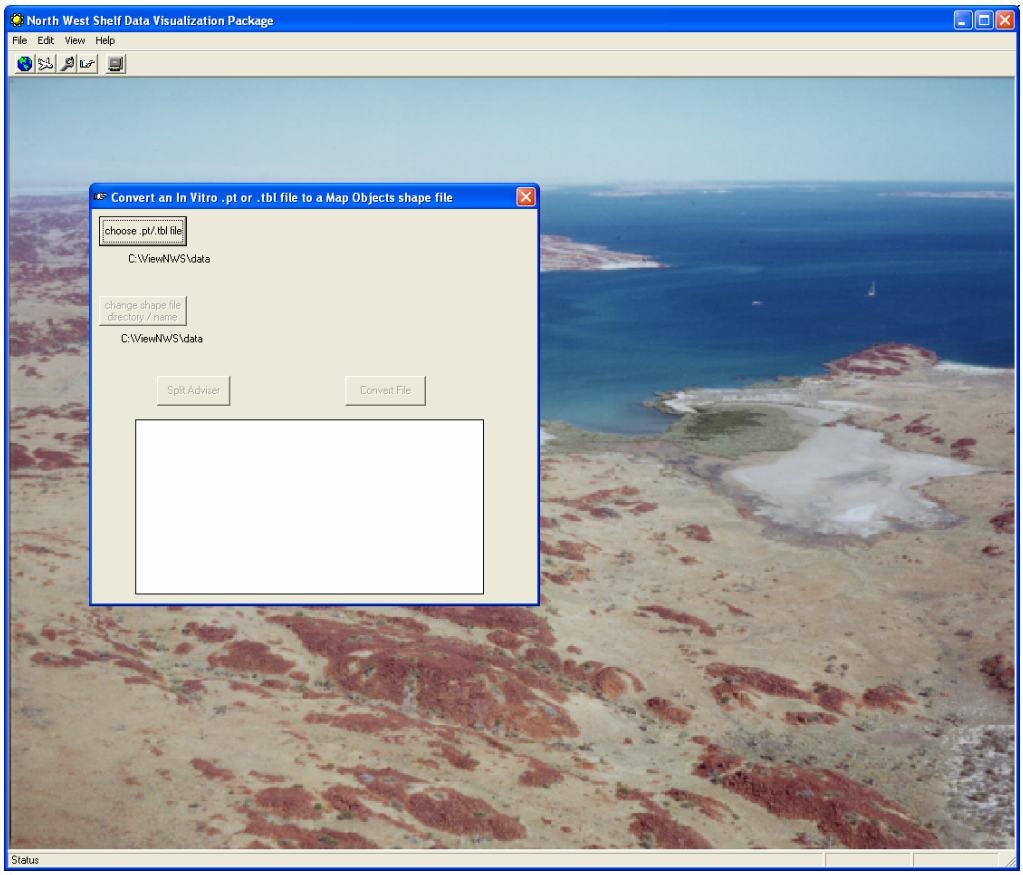


Figure 2.6.40: Choose .pt/.tbl button.

Select the model output file that is to be converted to a shape file (figure 2.6.41).

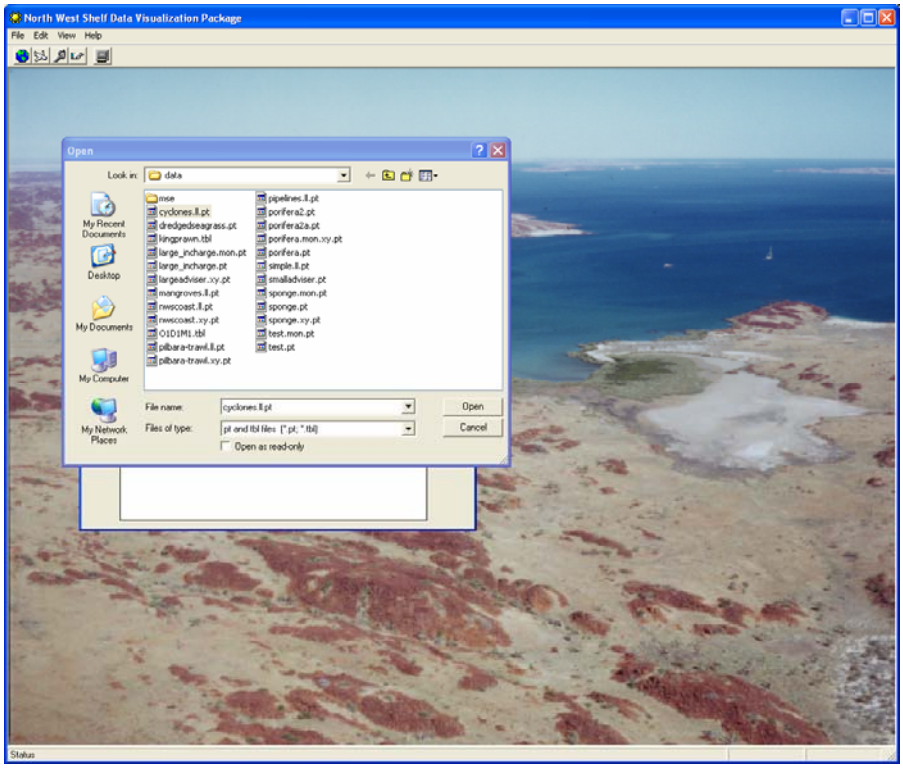


Figure 2.6.41: Selecting the .pt/.tbl file.

Next click **change shape file directory/name** and navigate to where the converted file is to be saved. Change the file name if required (figure 2.6.42).

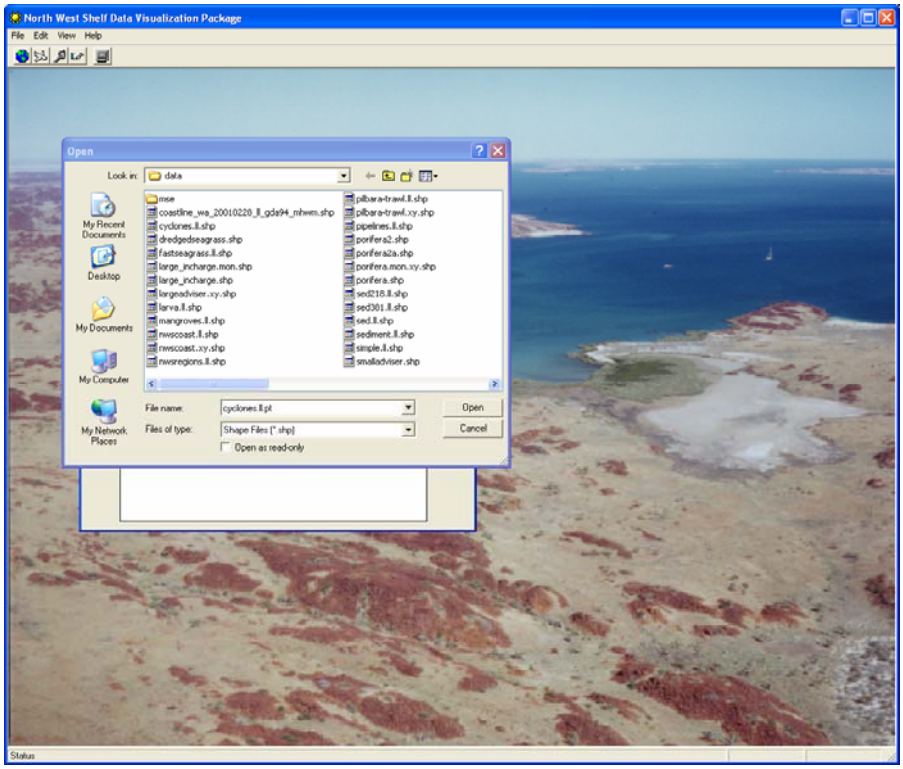


Figure 2.6.42: Selecting the .shp file location.

To check to see if the file selected is able to be converted to a shape file, click on **Split Adviser** Split Adviser.

To convert the pt./tbl file to a shape file click **Convert File** Convert File. A summary graphic of the file will be displayed (figure 2.6.43).

NOTE: Some of the .pt/.tbl files are very large. The process of converting them to shape files may take some time.

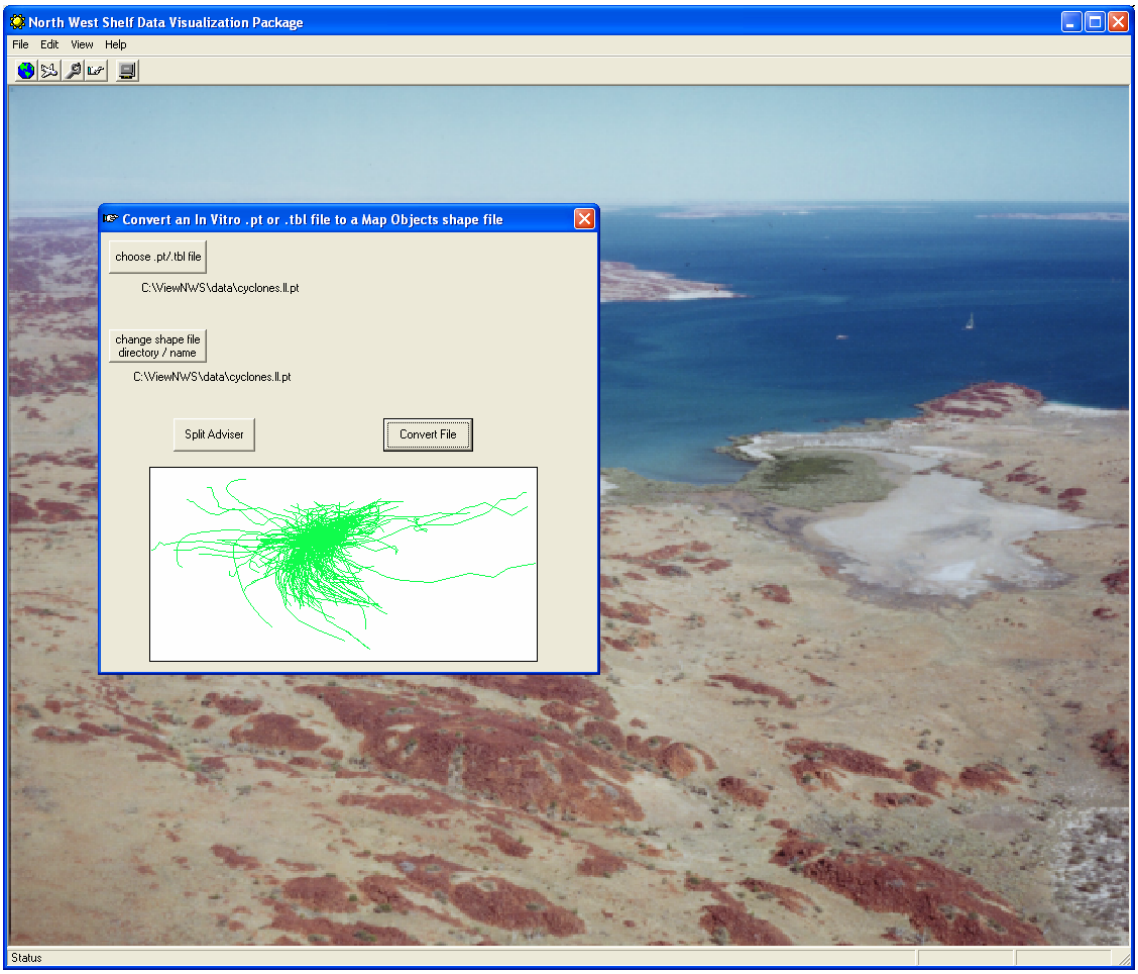


Figure 2.6.43: Converted file summary.

2.6.11 PowerPoint presentation

To run a *PowerPoint* presentation, the *PowerPoint* file *must* be placed in the 'data' directory. Failure to place it here will result in the presentation not running.

The next step is to open the setup screen from the drop down menu (see section 2.6.9 for details on opening the setup screen), click on the *PowerPoint* file button and navigate to the file. Remember to click on **Save Changes** Save Changes before exiting the setup screen (figure 2.6.44).

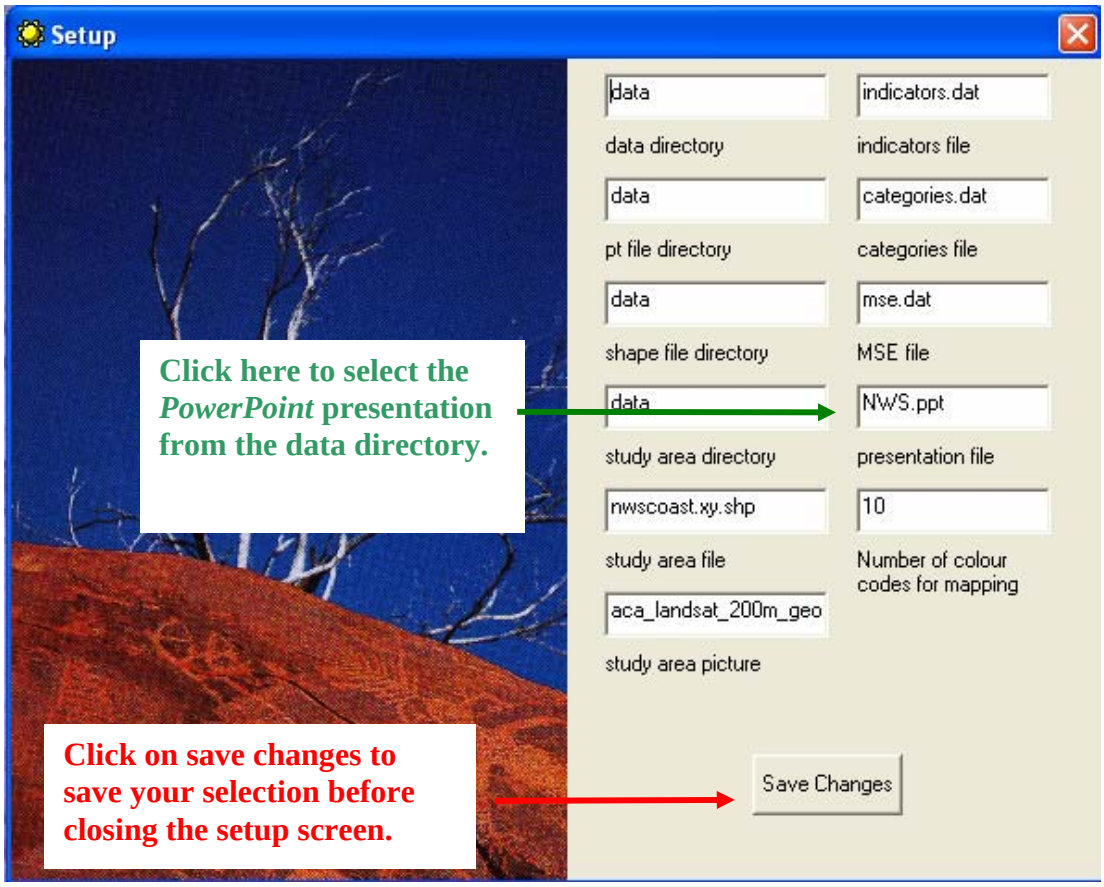


Figure 2.6.44: Setting up a *PowerPoint* Presentation.

To run the presentation click on the **View *PowerPoint* Presentation** button (figure 2.6.45) and double click on the image to begin presentation.

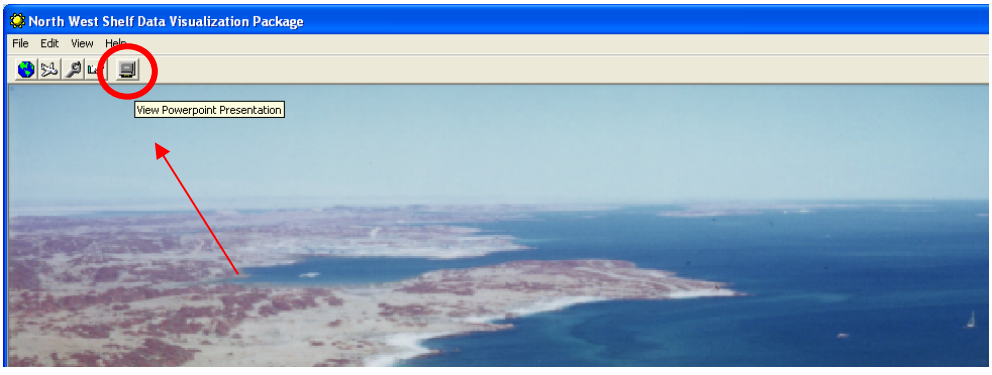


Figure 2.6.45: *PowerPoint* button.

2.6.12 Browser connection

ViewNWS has a browser link to the North West Shelf Study home page.

To access, click on **View**, **NWS home page** (figure 2.6.46) and a browser window will launch with the North West Shelf home page (figure 2.6.47).

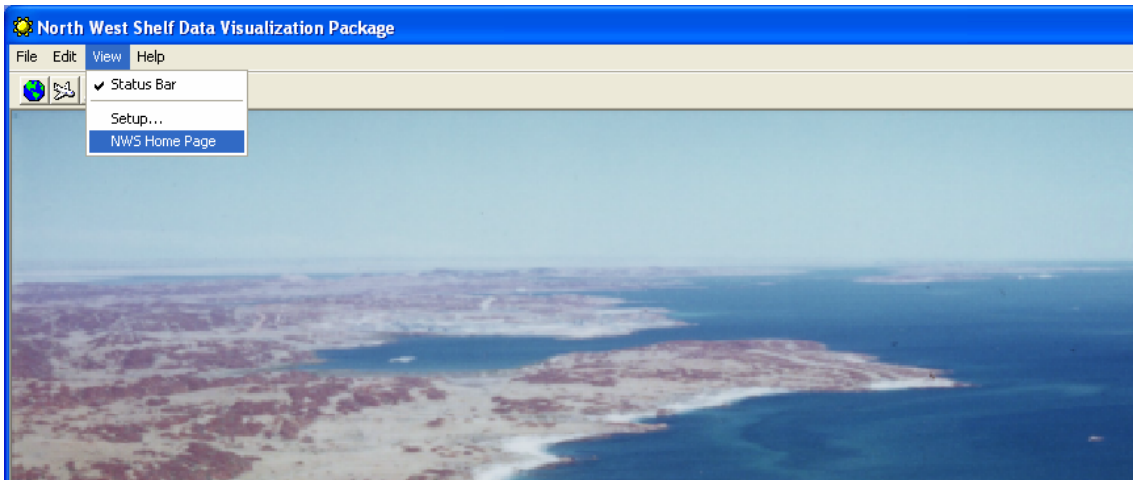


Figure 2.6.46: *ViewNWS* browser.

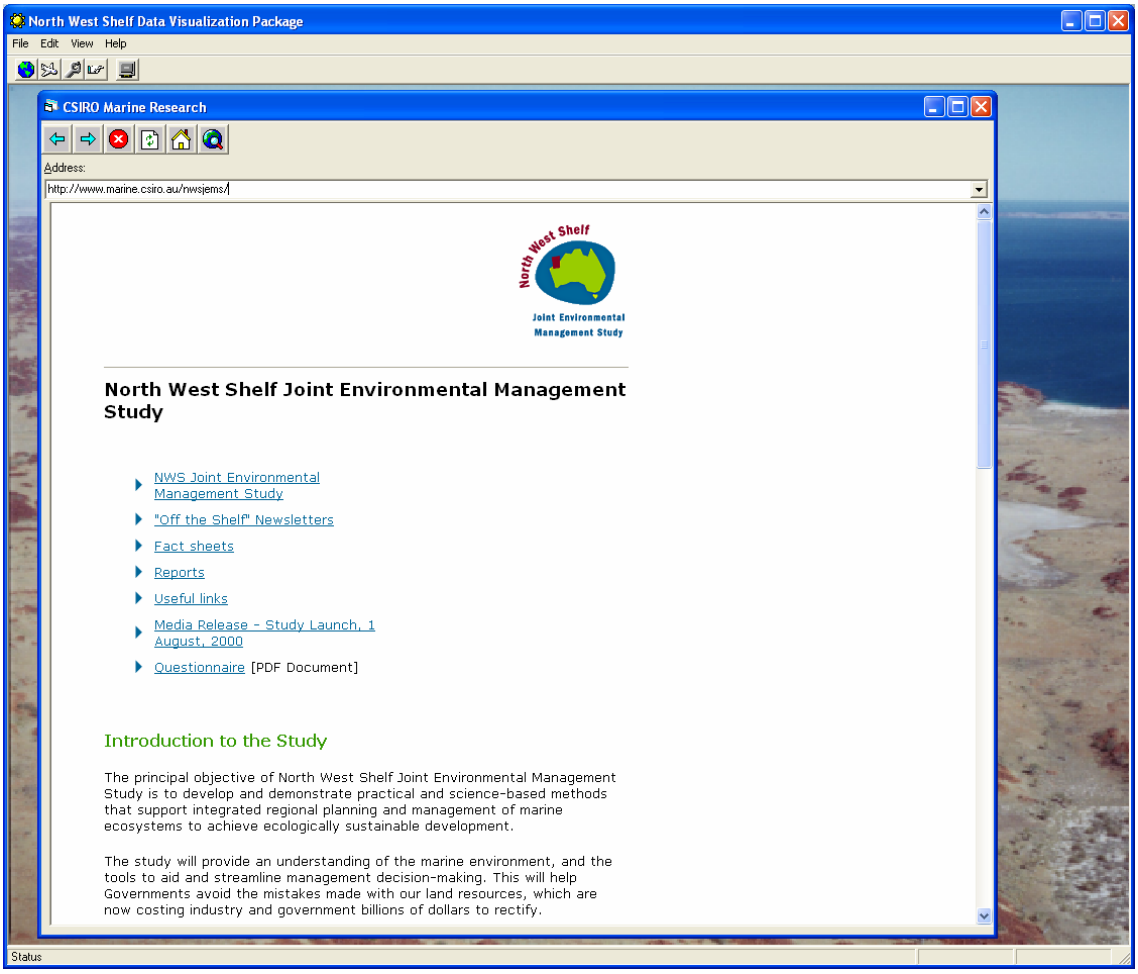


Figure 2.6.47: North West Shelf Study home page.

2.6.13 Help menu

The *ViewNWS* user guide is accessed by selecting **Help**, from the **Help** menu (figure 2.6.48).

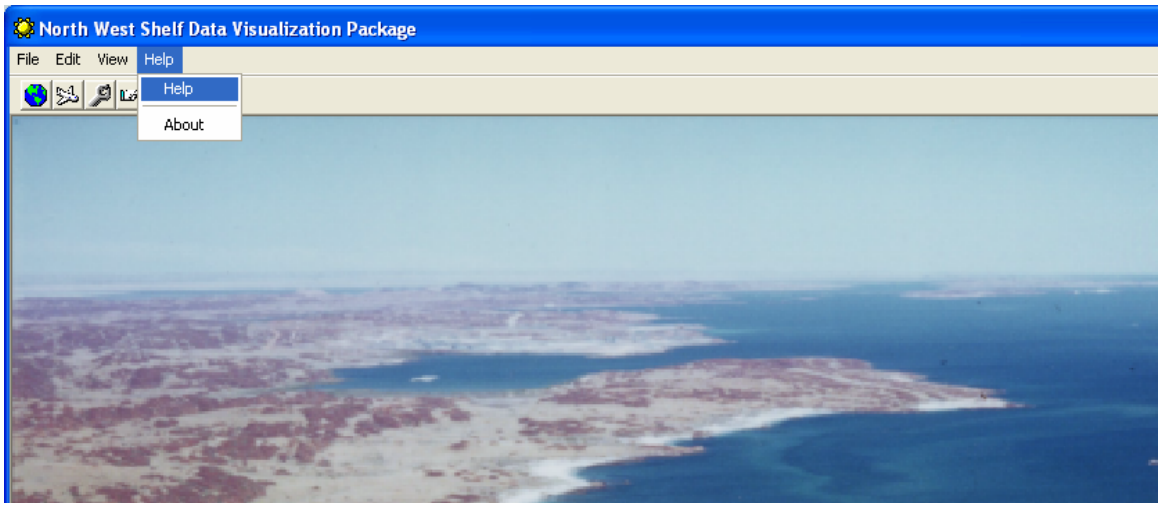


Figure 2.6.48: Help menu.

System information about *ViewNWS* is accessed by selecting **About**, from the **Help** menu.

3. PROGRAMMING DOCUMENTATION

Outline of Forms used in ViewNWS

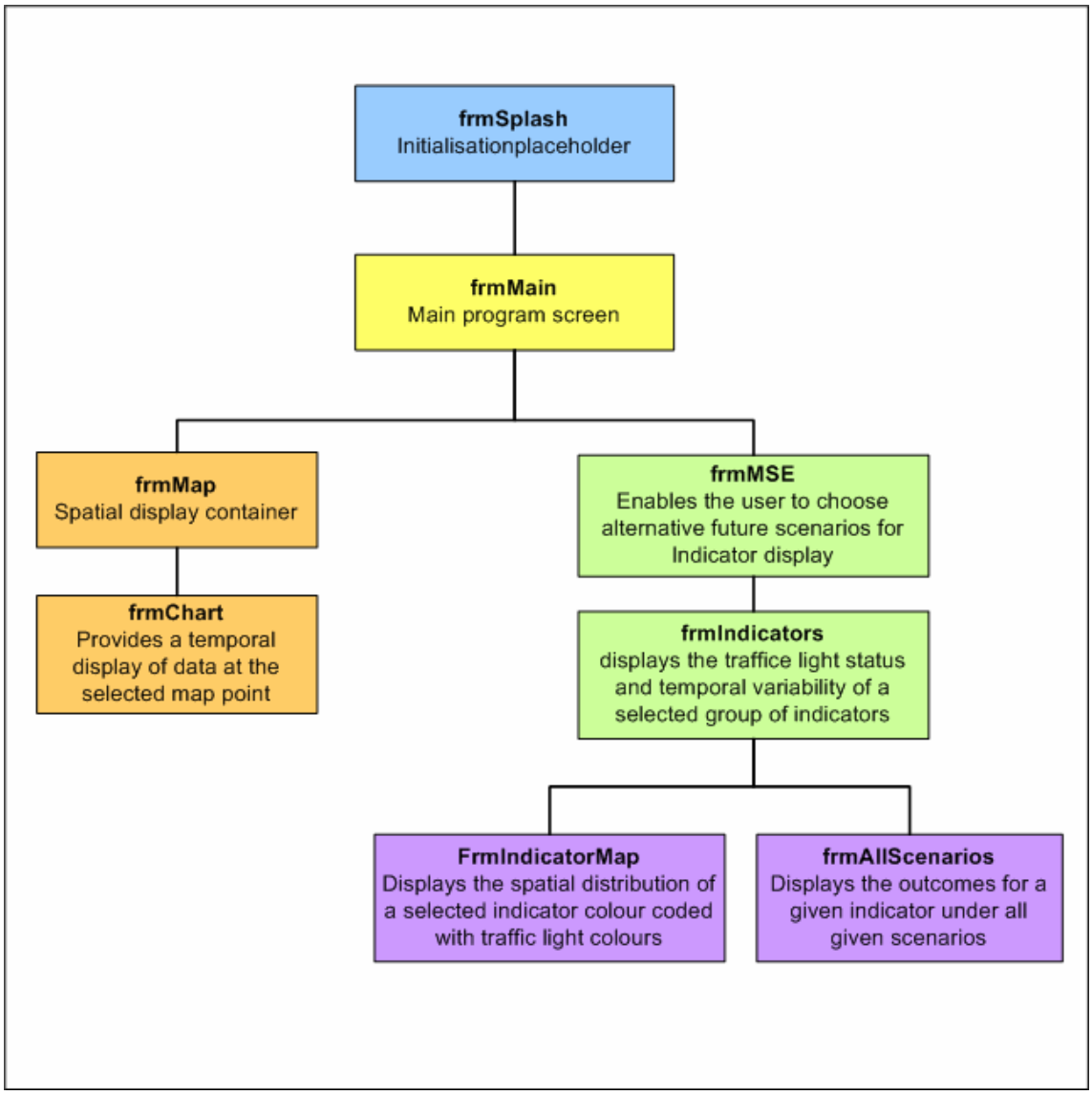


Figure 3.1: Main program screens.

3.1 Main program screens

ViewNWS is a Multi-Document Interface program, the functional parts of which are spawned from a main window that serves as a container for the other windows. As such, the most effective way to describe the program is by featuring the windows that the user sees when operating the program:

Form Splash:	As with most splash screens it acts as a visual placeholder while the program loads data files for the initial display. This is not a trivial function – it discourages the user from continually double clicking the application because they are under the misapprehension that the application is not loading.
Form Main:	Main control screen (below). This screen is the Multi-Document container that allows access to the main program functionality. Spatial data viewing can be undertaking from this screen by loading the map display.
Form Map:	Accessed from Form Main. A screen that allows the display of spatial map data. The basic map format is as shape files, but image layers (.bil files), and ArcInfo coverage (.adf files) may also be added. The map images can either be printed directly to a printer or exported either through the clipboard or saved as JPEG or BMP files. The implementation will use a MapObjects ActiveX component to display the maps, with legends and scale bars derived from the imported data. Form Map can be launched in two different styles. With the first, a map of the NWS forms the base map, in the second a composite satellite image will form the base.
Form MSE:	Accessed from Form Main. Enables the user to select which model scenario is used when displaying a set of indicators. This is defined through the 3 by 3 by 3 matrix defined above. At present, it is envisaged that the choices will be selected through pull down menus with a rich-text display giving a brief outline of each option.
Form Indicators:	Accessed from Form MSE. This is the main “traffic light” screen. It shows all, or a grouping of, indicators with their associated traffic lights. The value of the thresholds which determine the colour of the traffic light can be set by entering in values in the corresponding boxes on screen.
Form IndicatorMap:	Accessed from Form Indicators. Displays how an indicator is spatially distributed. The spatial indicator data is displayed in the same manner as the traffic lights i.e. using green, yellow and red. This form is implemented using the *MapObjects* component.
Form AllScenarios:	Accessed from Form Indicators. Uses traffic lights to display how an indicator performs across all included scenarios.
Form Chart:	Accessed from Form Indicators. Displays how an indicator performs over time. This form is implemented using the *Gigasoft ProEssentials* charting component.

3.2 Ancillary functionality

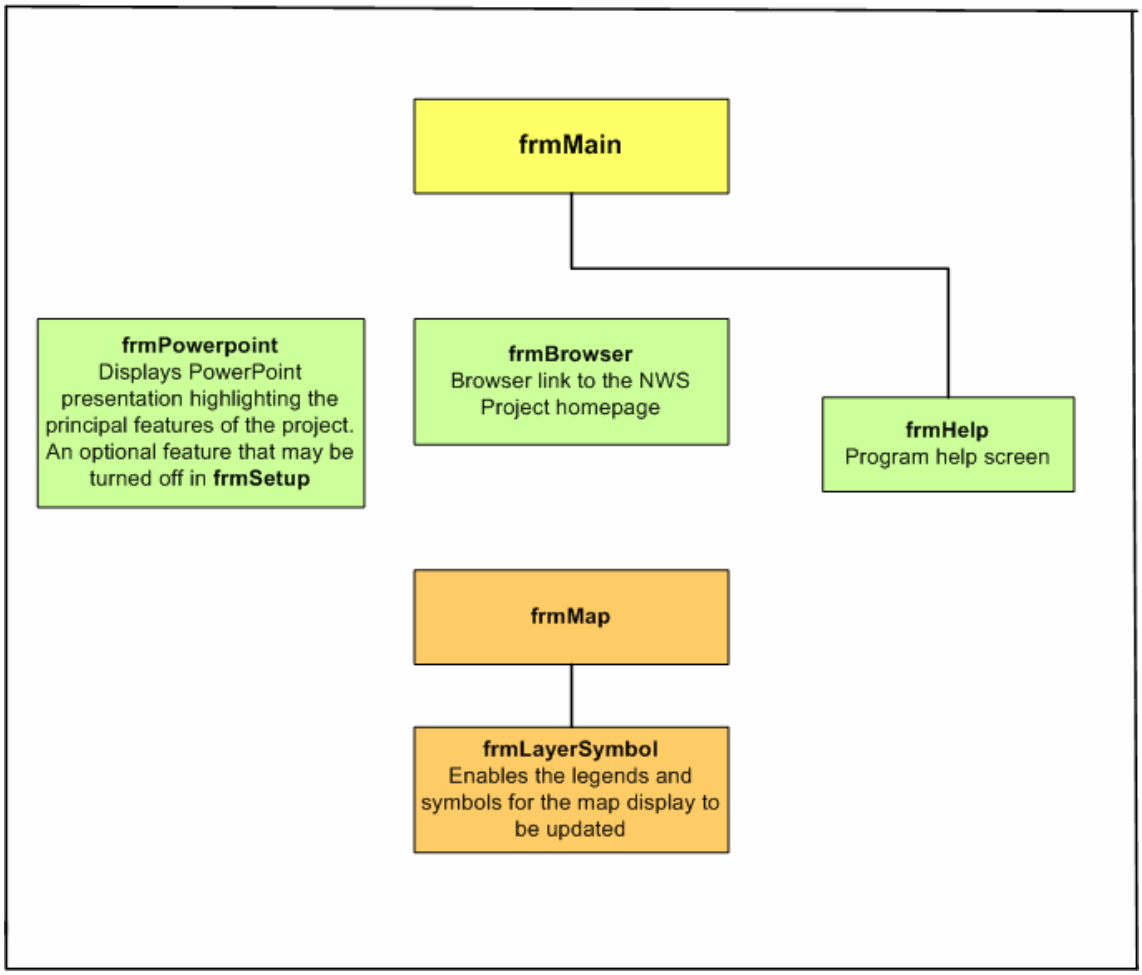


Figure 3.2.1: Ancillary screens.

3.2.1 Ancillary screens

Form Help:	Accessed from Form Main. Help display formatted in HTML
Form Powerpoint:	Accessed from Form Main. A container for a *PowerPoint* presentation that can be run within the application.
Form Browser:	Accessed from Form Main. A browser link to the project's home page.
Form LayerSymbol:	Accessed from Form Map. The "Layer Symbol" form allows the editing of layer attributes (such as colour and symbols) within a map layer.

3.2.2 Class modules

Class modules are code containers and are not seen by the user.

Class Startup:	Reads in general program data such as indicators and initialises global variables.
Class Declarations:	Provides a place for declaring global constants used by the program.
Class PluginDeclarations:	Provides a place for declaring global constants used by the *MapObjects* and *Gigasoft ProEssentials ActiveX* components.

3.3 Description of functions used in ViewNWS

ViewNWS uses functions to call on a variety of subroutines to perform various tasks.

3.3.1 Module startup

Sub Main()
Startup routine

Loads Splash Screen frmSplash
Calls ReadIniFile to load initialisation file
Performs initial connection to the database as BasicData.Connection1
Calls LoadIndicatorsDatabase
Calls LoadCategoriesDatabase
Loads Main program screen frmMain
Unloads Splash Screen

Public Sub LoadIndicatorsDatabase()
Loads the indicator descriptions table into the indicator array Indicator(Number of indicators)

```
Private Type Indicator
   Name As String
   Summary As String
   AdviserName As String
   IndicatorFieldName As String
   TargetThreshold As Double
   LimitThreshold As Double
   Units As String
   CategoryID As Integer
End Type
```

Public Function LoadCategoriesDatabase()
Loads the category list table into the Category(number of categories) array.

```
Public Type Category
   Category As String
   NumIndicators As Integer
```

Indicator(MaxIndicators) As Integer
End Type

Private Sub ReadIniFile()
Reads the file Esturine.ini into “Setup” data type

Public Type SetupFiles
ApplicationTitle As String
ApplicationTitle_desc As String

data_directory As String
data_directory_desc As String
shape_file_directory As String
shape_file_directory_desc As String
study_area_dir As String
study_area_dir_desc As String
study_area_file As String
study_area_file_desc As String
study_area_picture As String
study_area_picture_desc As String
slideShowDocument As String
slideShowDocument_desc As String
ProjectHomePage As String
ProjectHomePage_desc As String
AccessLocation As String
AccessLocationDesc As String
BreakCount As Integer
BreakCount_desc As String
End Type

Public Sub SaveIniFile()
Creates a backup copy of the initialisation file called “ViewNWS.bak” then saves the setup array to the file “ViewNWS.ini”

3.3.2 frmMain

Private Sub MDIForm_Load()
Performs initial load of the MDI form

Private Sub MDIForm_Unload(cancel As Integer)
Saves the application position for next time it is used

Private Sub mnuFileExit_Click()
Exits the program

Private Sub mnuHelpOpen_Click()
Launches frmBrowser with the HTML help file

Private Sub mnuHelpAbout_Click()
Launches the “About” dialog frmAbout

Private Sub mnuViewWebBrowser_Click()
Launches frmWebBrowser with the project home page

Private Sub mnuViewStatusBar_Click()
Shows and hides the staus bar

Private Sub tbToolBar_ButtonClick(ByVal Button As MSComctlLib.Button)
Launches functions from the toolbar. Available functions are:
View PowerPoint Presentation
Load Map(“Map”)
Load Map ("Photo")
Import data
LoadMSE

Private Sub LoadMap(BaseType As String) 'Map = map Photo = aerial photograph)
Loads the map child form frmMap and passes the form the information on whether it’s loading a map or image base

Private Sub LoadData()
Loads frmLoadData which controls the data conversion programs

Private Sub LoadMSE()
Loads the management strategy evaluation form “frmMSE”

Private Sub LoadPowerpoint()
Loads the *PowerPoint* presentation form frmPowerpoint

3.3.3 frmMap

Private Sub Form_Load()
Undertakes form loading duties
Calls InitialiseMap
Calls PointTo

Private Sub InitialiseMap()
Connects to the relevant geodata set and loads the base map or image
Calls Form_Resize

Private Sub Form_Resize()
When the form is moved and changed in size this function places the controls in the correct positions
Calls DoScalebar

Private Sub DoScalebar()
Calculates the map extents and then recalibrates the scale bar appropriately

Private Sub AddLayer()
Adds a layer to the base map
Opens a file open dialog to find the shape file
Connects to the relevant geodata set
Adds the layer to the map
Populates the fields combo box comboSelectField
Populates the minimum date combo box comboSelectMinDate
Populates the maximum date combo box comboSelectMaxDate
Renders the map colours using MapObjects2.ClassBreaksRenderer

Private Sub comboSelectField_Click()
Selects the data field within the map geodata set which is to be displayed and then renders it on the map

Private Sub comboSelectMaxDate_click()
Selects the end date for a temporal display loop

Private Sub comboSelectMinDate_Click()
Selects the start date for a temporal display loop

Private Sub cmdDisplayloop_Click()
Initiates or cancels the display of field data sequentially across time
Uses DateStart derived from comboSelectMinDate
Uses DateFinish derived from comboSelectMaxDate
Enables or disables the timer “timer1”

Private Sub Timer1_Timer()
Displays field data sequentially across time
Static variables: CurrentIndex, LastIndex
Calls ShowRange

Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
Launches the functions available from the map toolbar
Available functions are:
PrintMap
CopyClipboard

AddLayer
RemoveLayer
PointTo
measure
ZoomRect
"RestoreFull" : Restores map extent to the original.
ZoomIn
ZoomOut
Pan
ShowHide
cmdDisplayloop_Click
EditLegend

Private Sub PrintMap()
Opens a print dialog box CommonDialog1 and sends the current map image to the printer.

Private Sub PointTo()
Changes the mouse pointer to an arrow icon

Private Sub measure()
Changes the mouse pointer to a pencil icon

Private Sub ZoomRect()
Changes the mouse pointer to a zoom icon

Private Sub ZoomIn()
Changes the mouse pointer to a zoom in icon

Private Sub ZoomOut()
Changes the mouse pointer to a zoom out icon

Private Sub Pan()
Changes the mouse pointer to a pan icon

Private Sub CopyClipboard()
Exports the current map as a bmp file on the clipboard

Private Sub EditLegend()
Launches the legend editor frmLegend and tells it which layer to edit

Private Sub ShowHide()
Hides or reveals the various buttons on the form

Public Sub ShowRange()
Displays field data sequentially across time

Private Sub legend1_LayerDblClick(Index As Integer)
Updates the legend and map layers.

Private Sub Map1_AfterTrackingLayerDraw(ByVal hDC As StdOle.OLE_HANDLE)
Measures the distance on the map when the measuring tool is selected.

Private Sub Map1_DragDrop(Source As Control, X As Single, Y As Single)
Provides movement for the pan tool

Private Sub Map1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
Provides functions dependent on the shape of the mouse cursor over the map
moPencil does measurement
moPan does pan
moZoom does area selection zoom
moZoomIn does zoom in
moZoomOut does zoom out
moArrow launches the time series chart for the data point

Private Sub txtTimer_Change()
Changes the time interval between layer refreshes for the temporal display

3.3.4 frmChart

Loaded when a data point is clicked on the map; displays the temporal data distribution in a chart

Public variables:
ChartX() As Double
ChartY() As Double
ThisLayer As String

Private Sub Form_Load()
Loads the form
Calls ShowPointHistory
Calls SetupChart

Private Sub ShowPointHistory()
Queries the map geodata set and obtains the data

Private Sub SetupChart()
Places the data in a chart control

3.3.5 frmMSE

Allows the selection of the management strategy evaluation combination

Public variables:
CurrentOperatingModel As Integer
CurrentDevelopmentScenario As Integer
CurrentManagementStratagy As Integer

Private Sub Form_Load()
Loads the form and places the controls
Calls ReadMSEstartupDataBase
Calls ReadIndicatorsDatabase

Private Sub ReadMSEstartupDataBase()
Reads the overall database attributes

Private Sub ReadIndicatorsDatabase()
Reads the specific indicator data from the database

Private Sub cmdIndicators_Click()
Opens the indicators form frmIndicator

Private Sub cmdSetupCategories_Click()
Launches the database for editing

Private Sub comboDevelopmentScenarios_click()
Chooses the relevant development scenario

Private Sub comboManagementStrategies_click()
Chooses the relevant management strategy

Private Sub comboOperatingModel_Click()
Chooses the relevant operating model

3.3.6 frmIndicators

Public variables:
CurrentChart As Integer
OldChart As Integer
MapToggle As Boolean 'Map = true indicator = false
SelectedCategory As Integer
OperatingModel As Integer
DevelopmentScenario As Integer
ManagementStratagy As Integer

Private Sub Form_Load()
Loads form and places controls

Calls AddCategories
Calls listCategory_Click

Private Sub listCategory_Click()
Chooses the category of indicators which is to be displayed
Calls SetupPlots

Private Sub SetupPlots()
Places indicator data into the charts
Calls SetTrafficLight

Private Sub SetTrafficLight(WhichChart As Integer)
Sets the traffic light colour

Private Sub cmdChangeThresholds_Click()
Changes the target and limit thresholds for the selected indicator
Calls GrowMe

Private Sub cmdMapIndicatorMode_Click()
Chooses whether the “all scenarios” or “indicator map” screen is selected when the indicator icon is clicked

Private Sub cmdSaveThresholds_Click()
Saves the changed threshold

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
Highlights the selected chart

Private Sub Form_Unload(cancel As Integer)
Provides unload services

Private Sub Pego1_MouseDown(Index As Integer, Button As Integer, Shift As Integer, X As Single, Y As Single)
If the thresholds mode is selected then this function selects the indicators for which the thresholds are to be changed; if not then it makes the selected chart full screen
Calls MoveBox (Index)
Calls GrowMe (Index)

Private Sub Pego1_MouseMove(Index As Integer, Button As Integer, Shift As Integer, X As Single, Y As Single)
Initiates moving the highlighting box to the selected chart

Private Sub MoveBox(Index As Integer)
Moves the highlighting box to the selected chart

Private Sub picIndicator_Click(Index As Integer)
If the traffic light is displayed clicking the picture launches the ”all scenarios” form, frmAllScenarios; if the globe is diplayed clicking the pictures launches the “indicator chart” form, frmIndicatorChart

Private Sub GrowMe(Index As Integer)
Enlarges the selected chart so that it can be seen

Private Sub SaveThresholds()
Saves the changed thresholds to the database

Private Sub AddCategories()
Adds the categories to the categories list

3.3.7 frmIndicatorMap

Displays the spatial distribution for the selected indicator
Public variables:
FieldName As String
CurrentIndicator As Integer

Private Sub Form_Load()
Loads the form
Calls AddIndicatorLayer
Calls LoadBaseMap
Calls comboSelectDate.SetFocus

Private Sub LoadBaseMap()
Connects to the geodata set ans loads the underlying map

Private Sub AddIndicatorLayer()
Adds the selected indicator data layer to the map
Calls AddDates
End Sub

Public Sub UpdateIndicatorLayer()
Renders the map and sets the legend

Private Sub AddDates()
Adds the dates to the control comboSelectDates

Private Sub cmdCloseMe_Click()
Unloads the form

Private Sub comboSelectDate_click()
Selects the date to be displayed

3.3.8 frmAllScenarios

Show all possible scenario combinations for a given indicator
Public variables:
OldChart As Integer
ActiveCategory As Integer
ActiveIndicator As Integer
TheIndex() As Integer

Private Sub Form_Load()
Loads the form
Calls SetupForm
Calls SetupLabels
Calls ShowButtons
Calls SetupPlots

Private Sub SetupForm()
Places the controls on the form

Private Sub SetupLabels()
Sets the labels on the form

Private Sub SetupPlots()
Loads indicator data into the charts and sets the traffic light colours

Private Sub ShowButtons()
Decides if there is actually data to display for a given scenario combination

Private Sub cmdIndicator_Click(Index As Integer)
Switches the displayed chart to that for the selected scenario combination

Private Sub Timer1_Timer()
Provides a delay so the labels can be placed without the program crashing

4. TECHNICAL USER INTERFACE

4.1 Introduction

The *NWS Technical User Interface* is a visualisation system written for Java 1.4. It uses a multiple document mechanism that allows multiple windows of the same or differing types to be displayed simultaneously. Data set types which may be displayed include spatial, time-series, image and table data. Attributes of a data set may be displayed in a variety of ways which allows a comparison of data either between data sets or within a data set itself. A recursive decent parser is implemented which provides analytical and mathematical functions for data manipulation or exploration. Data sets may be loaded from multiple sources with the two most common being files and variables generated by the parser. It is aimed squarely at the scientific user to aid in model debugging and analysis.

4.2 Data set types

The user interface currently supports the loading of spatial files in PT format, image files in PXM format and table files in TBL format. Time Series data may be integrated with these data sets and is automatically enabled. Tentative support for spatial data in ESRI Shapefile format and table files in DBF format is available, however this functionality has not been extensively tested.

4.2.1 PT file format

The interface supports a subset of the ASCII PT file format which is fully documented in Appendix B. Object classes supported are *Point*, *Polyline* and *Polygon* although only one class is supported per data set. Each object within a data set is required to have consistent attributes in both length and order. Vectors are required to be line delimited and although read are ignored and not available for use.

All PT attributes with the exception of *time*, which is converted to a time type, are read in as strings. When used as mathematical expressions it will be necessary to convert strings to numbers via a *num()* call.

4.2.2 PXM file format

For a full description see Appendix C. The interface supports all currently documented PXM formats including stacks (currently only the first image pf a stack is available for use) and lists. Raw format floating point images may only be read on same architecture machines. Attributes are extracted from the comments and if available provide geo-referencing information. The interface currently stores all images in double precision and care is needed when loading multiple images that memory limitations are not exceeded. Image files in the interface can be drawn in spatial windows as velocities (currently only available during NWP project file loading).

4.2.3 TBL file format

TBL files consist of three header lines followed by space delimited ascii lines with one line corresponding to one row of the table. The first line in a TBL file contains the *TBL* keyword. The second line contains the names of the columns. The third line contains space separated characters which identify the data type of the column and may contain *F* for double precision numbers, *T* for time and *C* for a character strings.

4.2.4 Data set filters

Filters may be used for extracting segments from large data sets. They may be used to reduce the amount of data read when memory is limited or when the data sets span a large spatial area or temporal period. They are currently implemented for PT files with TBL file support pending.

4.3 Windows

Menus are available on both the main window and the individual data windows. Options available from the main window operate on either a global scale or on the currently selected/focused window. Data window menu options are specific to that window or the currently selected data sets within the window.

With one exception, when a window requires a different type to that native to the data set, each data set loaded has only one instance and modification of attributes, for example colour, will be visible (this may require a refresh) in all windows which are displaying that data set. For windows which require a specific type, an example of which is the rendering of TBL files in a geographical window or rendering spatial data in a time series window, data sets are *mutated* into an appropriate type for display. Once *mutated* the data set is considered an instance for that data set and type and may again be displayed multiple times in different windows. Mutated data sets may contain extra data beyond the data sets' usual attributes which is needed for it to be able to render appropriately. The *Dump Details* menu option lists the data set name, location, reference counts and type.

Some windows have a similar look and are similar in use, the most obvious being those windows with a data set list on the left and the data display on the right. The window split may be moved by dragging with the mouse.

Data sets in these windows are displayed in reverse order so that the top of the list will be rendered last and will be displayed on top of previously rendered data sets. Data set rendering order may be changed by dragging and dropping although the mouse must be over the legend for drag to work correctly. Data sets may also be dragged and dropped between windows although this is currently only supported for *Geographical* (section 4.4.6) and *TSGeo* (section 4.3.5) windows.

The checkbox can be used to disable the rendering of a data set. To *select* a data set use the alt or shift key whilst clicking the left mouse button over the data set title. Depending upon the particular type of window, data sets may then be removed, renamed, parameters modified, etc.

4.3.1 Main window

Printing is functional for most of the windows, however some versions of Java do not properly carry across the previous print settings or those used during print setup.

Project file saving does not yet save everything it should and in some instances saves too much. Project files saved should be considered a template and edited as appropriate. After quitting, a *nws.go.save* file is saved by the software which may be used to extract information such as geographical bounds of a window, data set order, colours etc. Alternatively use the *Save Project* option to create the file and insert the required output into the main project file. *Close* removes all windows and clears all variables and caches; however it is recommended to quit and start again as the Java interpreter may have surplus memory that it has failed to relinquish.

Load table requires a full NWP (North West Shelf Project File) style data set reference and is rarely used. This option loads a reference to the data set into the cache without opening a window so the rules (section 4.5) can use it without cluttering the desktop window. *Attach table* can be used to attach tables to data sets (similar to a database join) but is best left for advanced users. *Add Filter* adds a global data set filter which is used to subset the data read in by the interface. See section 4.4.4 for further details. *Dump details* dumps out the data set cache details with reference counts and types and dump variables dumps out variable names, types and for some variables the values. The last two menu options are mostly used during debugging and will be of limited value during normal use.

Options allows for the setting of global options, however it is mostly a placeholder for future implementations.

The *Model* menu connects to the model daemon and selects a model for queuing. It is currently in the prototype stage and doesn't yet allow the running of the model but will allow model selection as well as downloading model configuration files for editing provided the server daemon is configured and running correctly on the model host. This should be considered only a demonstration of one way to allow model run execution and management.

4.3.2 Geographical window

The Geographical window displays native spatial data (point, line, polygon) and georeferenced image files. It will also attempt to translate TBL files with an *x* and *y* column or attribute into a point data set. Data sets are loaded via one of the Add Data set menu options (figure 4.3.2) and added to the list on the left side of the window. Selected data sets may be removed or renamed, as well as hiding/unhiding the legend and setting individual data set parameters. The spatial bounds of the selected data sets may be used to **Reset** the display to the maximum extent of their spatial bounds. The **Reset All** option may be used to set the display to the maximum extent of all data sets listed for the window.

Zooming in is accomplished by selecting the **Zoom In** tool (figure 4.3.1) and then via a standard select and drag the bounding box. Moving may be done with the **Move** tool by dragging. Zooming out is with a fixed scale from the clicked point. The **Info** tool will print to stdout the clicked location.

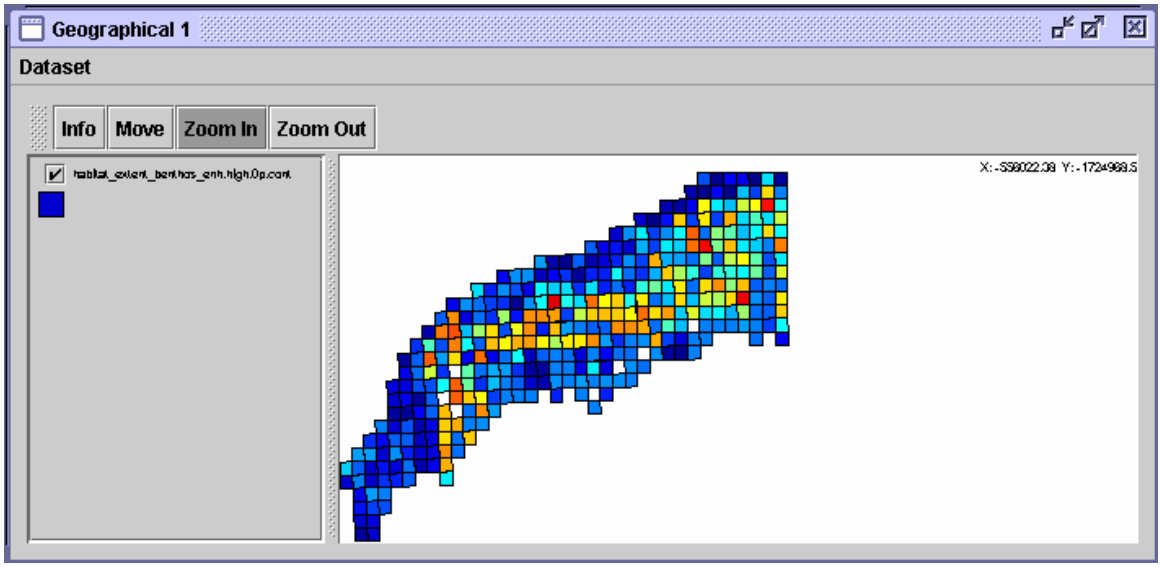


Figure 4.3.1: Geographical window with Zoom In.

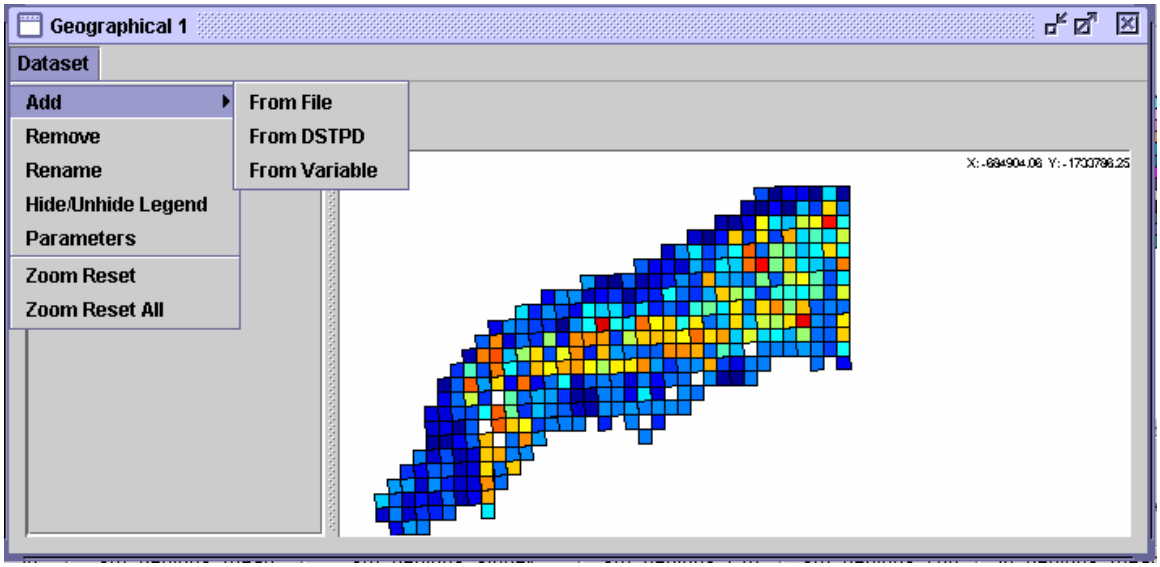


Figure 4.3.2: Geographical window with Add Dataset menu option

4.3.3 Time series window

Menus and options are similar to the Geographical window except that axes are displayed and all data sets are mutated into a time series type. Parameters which may be modified are the line colour as well as the Y axis attribute drawn. Although the X axis attribute is displayed it must be a time type and hence cannot be modified. The Y axis must be a numeric type. Point colour itself is usually coloured via *rules* described in section 4.4.8. Currently the parameter window buttons aren't functional and the window requires closing by clicking "x" in the top right corner. Unlike the Geographical window while zooming in, the scale is not maintained in X and Y and so a tall narrow zoom window will zoom in to exactly that area. Use zoom out with caution as it may not scale outwards as expected. If this occurs it may be necessary to use the *Reset* option and zoom in again.

4.3.4 Text window

A text window will open a text file for editing. This is mostly used with the prototype model daemon manager code to edit configuration files before sending to the manager for model run queuing. It may also be used to display items such as descriptions of strategies or scenarios.

4.3.5 Time Series Geographical window (TSGeo/GeoTS)

Similar to the Geographical except data sets with a time type attribute will be requested to render their data for a specific time. Data sets without any time attribute will be rendered as static and identical to how they would otherwise appear in the Geographical window. The data should be considered an instant snapshot, as (figure 4.3.3) opposed to an accumulation, unless the data set itself maintains the accumulations. Animation is an option although with multiple data sets or complicated data sets such as velocity vectors, rendering may never catch up to the scrollbar. Slower computers will have more of a problem. Resolution of the scrollbar may be an issue for data sets which span a large temporal period. The *Set Scroll Limits* option may be used to limit the scrollbar to a certain period and is entered as the two bounding seconds (standard UNIX time - seconds since 1/1/1970). Animate delay is the millisecond delay between scrollbar increments and the step is the actual number of seconds to increment. A function which uses the standard parser (see section 4.5) may be entered and will be evaluated for each redraw event. The variable *_time* is made available to the function and represents the current scrollbar time.

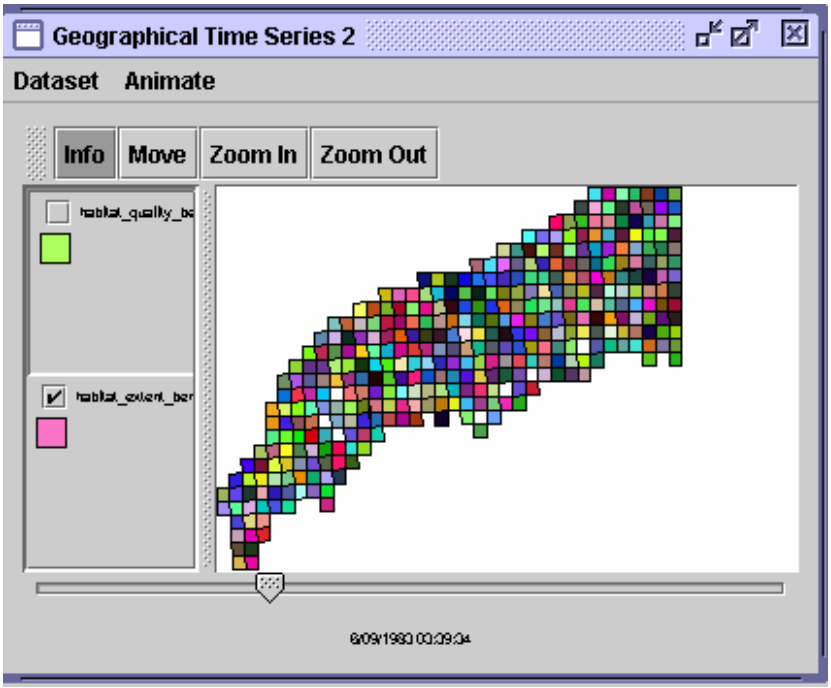


Figure 4.3.3: Geographical Time Series window.

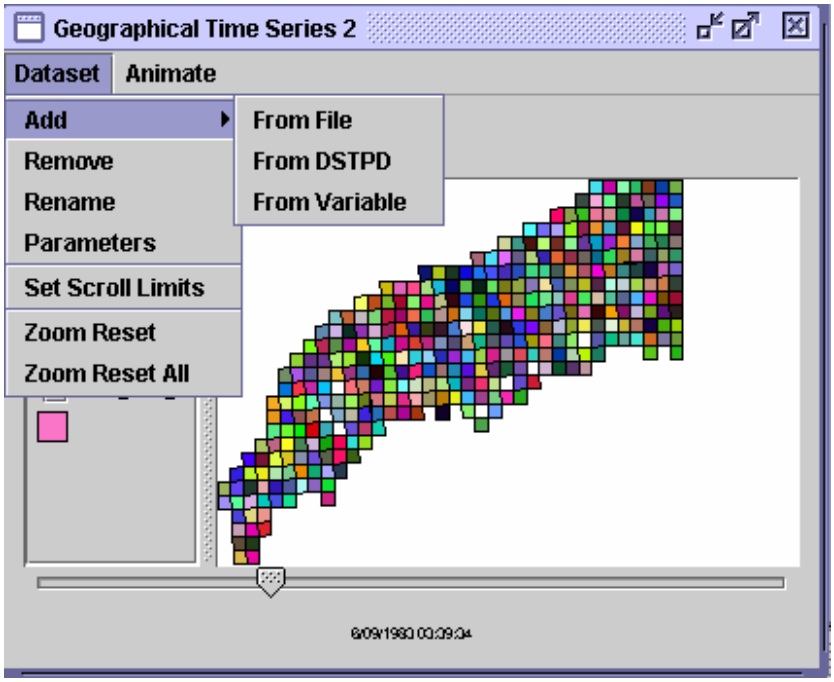


Figure 4.3.4: Geographical Time Series window with Add Dataset menu option.

4.3.6 Table window

Opens a TBL or DBF file as a table and displays the data in a similar manner to a spreadsheet (figure 4.3.5). This window will also display the non-spatial attributes of a PT file.

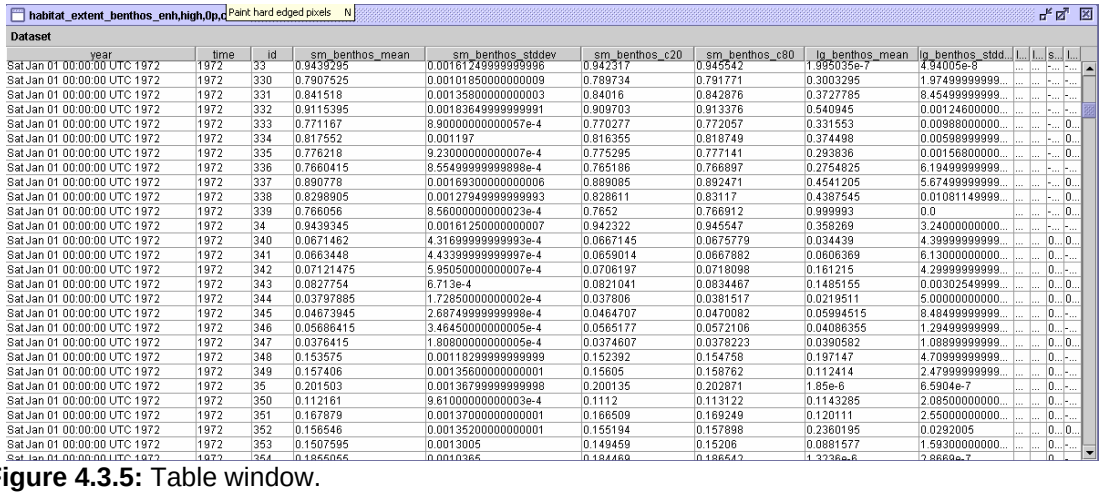


Figure 4.3.5: Table window.

4.3.7 Scoreboard window

The Scoreboard window is perhaps one of the most complex and needs a thorough understanding of both the expression parser and the data sets used. A Scoreboard looks similar to a spreadsheet although each cell is itself a function, which is evaluated by the parser. This can be a simple number, a colour or a complicated formula. Usually most of the configuration of this window is done via project files as typing formulas in a window quickly becomes tiring.

Each column is a strategy or scenario and each row represents an indicator. The variables *_col* and *_row* are made available to each cell - the *_col* is the column heading (usually the strategy or scenario name) and the *_row* variable is the result of parsing and executing the row function. Generally the row function is used with reference to *_col* to determine which strategy/scenario is applicable and then each cell value may be a reference to the *_row* variable (which has previously been evaluated for that column).

The pre-function may be used for anything although its normal use is to set a time variable or spatial extent variables which are used by the row functions to calculate the score for a particular time or area. Some of the parser functions may return cell values along with colours based on limits, fixed colours or some other formula. The limit setting is another option for the pre-function. The pre-function is pre-pended to each string for execution and evaluation by the parser and is not evaluated separately.

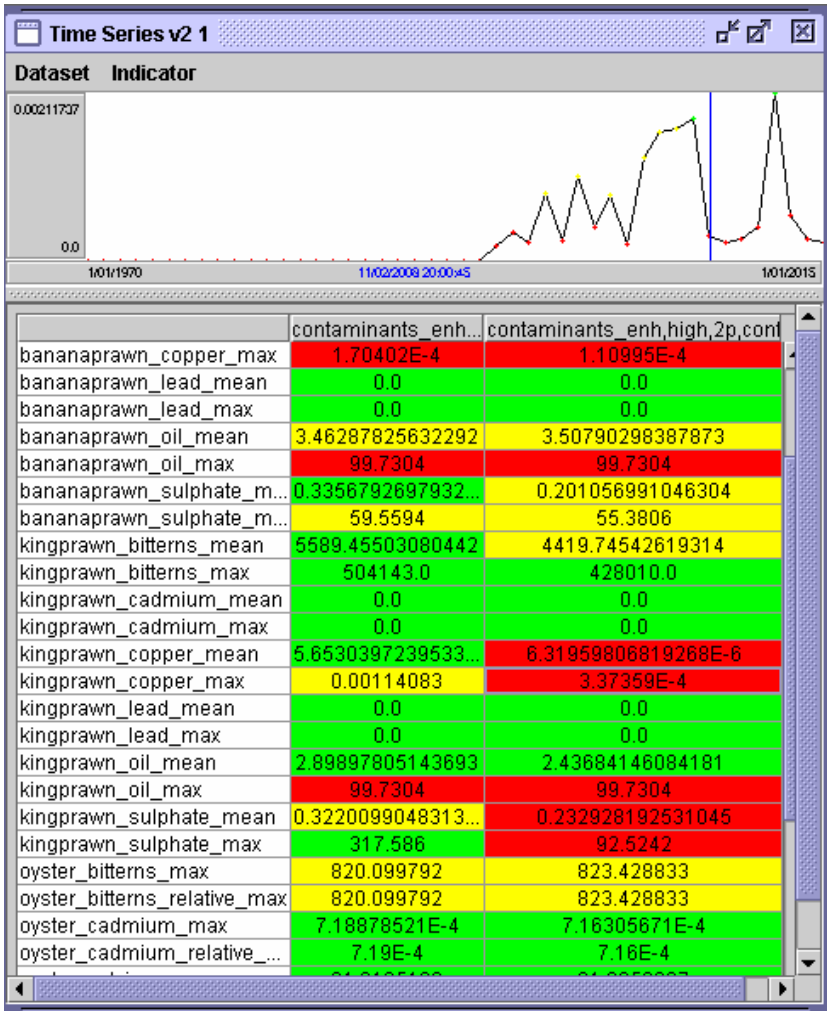
4.3.8 TS2 (Time Series 2) window

This window is a combination of both a scoreboard and a time series window (figure 4.3.6). Each table added is considered a strategy or scenario and a column is created. The rows represent each column within the table. Multiple tables may be added for multiple strategies/scenarios however the columns must be exactly the same in both name and type. Only number type data is displayed, however a time type must be included. The time series data displayed then comes from one of the selected cells (i.e. a column from the table) and the cell value represents the value at the time selected on the time series window. A left mouse click on the time series section will move the blue line and the new selected date/time will be displayed. The split, like other windows, may be moved by dragging with the mouse, however, unlike some other windows the time series section of the window will always display the whole data set and may not be zoomed or moved. The edit indicator option allows modification of the limits which define the colour of both the displayed time series data and the cell values for that indicator – it is consistent across all strategies/scenarios displayed. The inner/outer option determines the orientation, i.e. inside range is within either the desired or not desired values. The four values represent the thresholds, i.e. an inner type value will be considered:

- bad (red) if it is less than the first value or greater than the fourth value;
- yellow if between the first and second, or third and fourth; and
- okay (green) if between two and three.

The outer orientation is the opposite in colour. Due to the way values on a threshold boundary are selected, data on the boundary may not resolve to the desired colour. This is usually obvious when setting targets as opposed to desired ranges and there may be

some resolution issues between scrolling the bars and typing a value into the box. These are due to trying to make this window fairly simplistic but still usable without resorting to rule parsing.



	contaminants_enh...	contaminants_enh,high,2p,conf
bananaprawn_copper_max	1.70402E-4	1.10995E-4
bananaprawn_lead_mean	0.0	0.0
bananaprawn_lead_max	0.0	0.0
bananaprawn_oil_mean	3.46287825632292	3.50790298387873
bananaprawn_oil_max	99.7304	99.7304
bananaprawn_sulphate_m...	0.3356792697932...	0.201056991046304
bananaprawn_sulphate_m...	59.5594	55.3806
kingprawn_bitterns_mean	5589.45503080442	4419.74542619314
kingprawn_bitterns_max	504143.0	428010.0
kingprawn_cadmium_mean	0.0	0.0
kingprawn_cadmium_max	0.0	0.0
kingprawn_copper_mean	5.6530397239533...	6.31959806819268E-6
kingprawn_copper_max	0.00114083	3.37359E-4
kingprawn_lead_mean	0.0	0.0
kingprawn_lead_max	0.0	0.0
kingprawn_oil_mean	2.89897805143693	2.43684146084181
kingprawn_oil_max	99.7304	99.7304
kingprawn_sulphate_mean	0.3220099048313...	0.232928192531045
kingprawn_sulphate_max	317.586	92.5242
oyster_bitterns_max	820.099792	823.428833
oyster_bitterns_relative_max	820.099792	823.428833
oyster_cadmium_max	7.18878521E-4	7.16305671E-4
oyster_cadmium_relative_...	7.19E-4	7.16E-4

Figure 4.3.6: Time Series 2 window.

4.3.9 XY window

Similar to the Times Series window except this window uses a number type for the X axis instead of a time type.

4.3.10 Histogram window

The Histogram window will bin XY data (single data set) into a histogram or will display continuous data as a filled in polygon. Two main graph types are implemented. The 2D version is binned along the X axis and the 3D version is sliced through X at regular intervals and each slice plotted as a continuous graph with the Z values interpolated. If specified, minimum and maximum values may be displayed on the 2D version. X axis values may be of number, time or string array types with the latter only available for 2D plotting. The expression parser is used for evaluation. The histbin() function is used for 2D numerical and temporal plotting, the strbin() function for 2D

string plotting and the tinslice() function used for 3D numerical and temporal plotting. The variables _x, _y and, if applicable, _z are mapped to the selected attributes. The variable _nb is mapped to the number of bins however for the 2D string plot the number of bins is determined by the unique values of the X variable.

4.4 North West Shelf Project Files (NWP)

4.4.1 NWP File format

The NWP File Format is a line delimited ASCII text file with the first line containing **`NWS Project File`** as the magic string. Commands consist of a keyword followed by a sequence of parameters. The keyword `new` is used to specify a new window with the following argument being the type of window. Some single data set windows allow extra parameters to be specified on this line although most are configured with keywords and parameters on the following lines. Comments may be entered by specifying a # as the first character of a line. Blank lines are allowed although ignored. Case is sensitive and all currently implemented keywords are lower case.

The commands are processed in a backwards hierarchical fashion so that commands which are unknown to the main window are passed to the last specified data window. If the data window is unable to process the command it is then, for example, passed to the last specified data set although the actual hierarchy will depend on the window and its requirements.

The NWP file is parsed twice on start-up. This allows data windows which have data sets loaded from variables, or generated with rules, a second chance to configure themselves for those instances where the data set specification is not available during instantiation. Commands which fail to be processed the second time are written, along with an error message, to the console.

4.4.2 Data set references

Data sets are referenced in a similar manner to a web page URL. The format of a data set reference is

```
<transfer type>://<hostname>/<location>[#<mutate type>]
```

Transfer Type is currently either var or file but in future may add dstp, ftp etc.

Hostname is currently always localhost.

Location is the pathname for a file or the variable name (with a $) for a transfer type of var.

Mutate type is usually blank but can be, for example, SSPOINT for a TBL file displayed in a Geographic window.

4.4.3 General window options

Although each window requires and allows for different configuration lines there are similarities between many of the windows. Window location, size and data set viewing bounds are usually listed first, followed by a list of data sets with their configurations.

Most of these are optional and if not specified default values will be used.

```
title <window title>
```

Title changes the current windows title

```
location <x> <y>
```

Location sets the location of the current window relative to the main window.

```
size <xsize> <ysize>
```

Size sets the size of the current window in pixels.

```
transform <1> <2> <3> <4> <5> <6>
```

Transform is a six number affine transform used to scale and translate the window. This is usually entered by zooming or moving to the desired location and extracting the numbers from the saved project file.

```
iconified
```

Iconified iconifies the window.

4.4.4 Setting global data set filters

The format of a filter line is:

```
filter <function>$<dataset_attribute>=<value>

eg      filter min$x=117.5
```

For PT files the function may be either min, max, equal, substring or string, the first three being numeric comparisons and the latter two being string comparisons. The attribute may be one of the attributes of the data set or one of *x, y, longitude, latitude* which is compared with the spatial range of each data record. For numeric comparisons if the value contains a decimal point the comparison will be done in double precision otherwise it will be done as two 64 bit long integers. Multiple filters may be used to limit the data set to a specific spatial range, temporal period and/or some other attribute based comparison (e.g. habitat type).

4.4.5 Pre-loading multiple TBL files

Pre-loading files allows multiple data sets of the specified type from a directory or location to be loaded with a single command. These data sets are not associated with any window although their reference count will always be at least one so they are available for use in expressions. Typically this command is used to load all the TBL data sets from a model run with one line. The downside to this command is that it may load data sets which are not used, resulting in excessive memory use or in the worst case load files with conflicting attributes which may cause some expressions to fail.

The format of the preload line is:

```
preload <dataset type> <dataset transfer type> <host>
<location/dir> <suffix>

e.g.
preload tbl file localhost /usr/roger/Run-20010807125926 tbl
```

4.4.6 Geographical window options

```
new geographical
```

Opens a new geographical window.

```
dataset <dataset reference>
```

Adds the data set to the window list.

The data set reference may be followed by a list of data set specific options which in the case of a geographic window are usually colour references. These are of the hexadecimal format *0xaarrggbb* where *aa* is the alpha value and *rr*, *gg*, *bb* are the red, green and blue values respectively. The data set reference may also be followed by `highlighted` if the data set is selected or `uncheck` if the data set is not to be displayed.

4.4.7 Geographical time series window options

```
new geotimeseries
```

Opens a new geographical time series window. Similar options to the geographic window, except there is an optional `function <expression>` which sets an expression which is evaluated for each time step. This option may be used to modify data set colours for geographic areas based on thresholds (see example in 4.7.6).

4.4.8 Rule options

```
new rule
```

Opens a new rule window although if not specified an empty one will be created by default.

Apart from the general window position commands, other optional commands are of the form:

```
rule <boolean expression>#<rule action>
```

In most instances the *boolean expression* will be simply true although false can be used to temporarily disable evaluation of complex/slow rules. Although now rarely used the expression can also be a boolean array in which case the action must evaluate to an array of the same dimensions which is then used in selective assignment. This latter option is depreciated and not recommended as there are now cleaner alternatives for handling array assignments (e.g. *ternary()*).

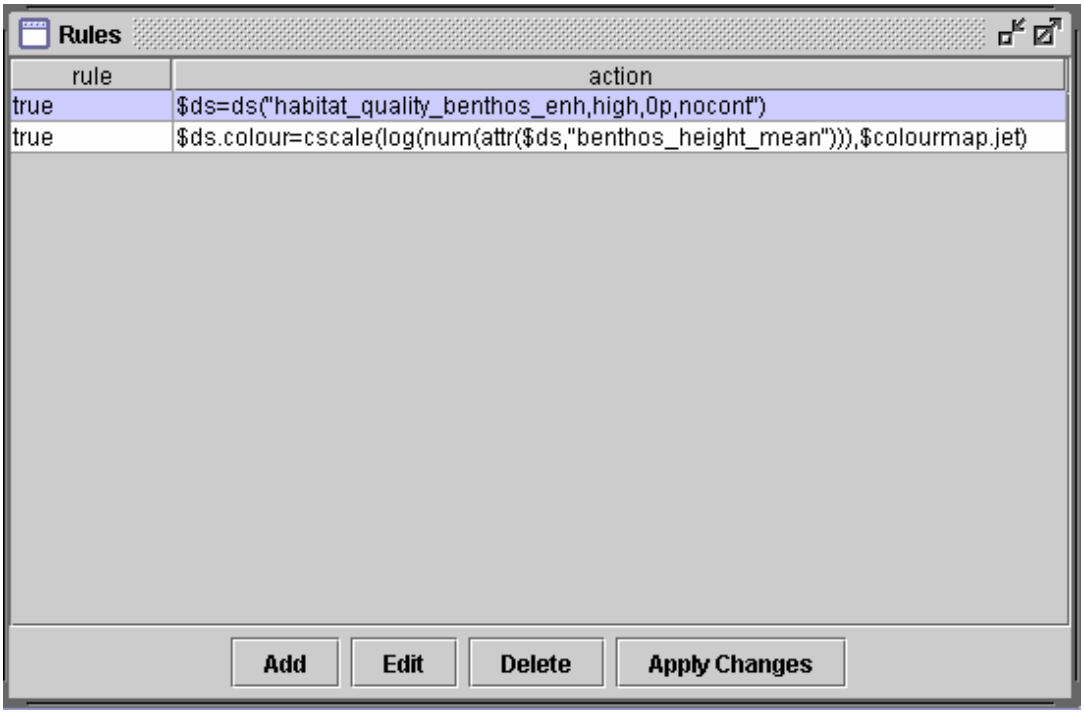


Figure 4.4.1: Rules window.

4.4.9 TBL loading

```
new tbl <dataset reference>
```

Loads a single tbl file without opening a window.

4.4.10 Table window options

```
new table [dataset reference]
```

Loads a single *TBL* file into window. The data set may be specified in a similar fashion to the previous non-window loading mechanism or alternatively a `data set <dataset reference>` command may be included.

4.4.11 Time series window options

```
new timeseries
```

Similar to the geographic window options. Data sets must be mutated to an SSTS type. The attributes used will default to the first *time* and *number* type in each data set specified and should be changed interactively if desired.

4.4.12 Scoreboard window options

Opens a new scoreboard window. Window size and location options are identical to previous windows. Specific commands consists of lines with the keyword `score`.

```
score col <space separated column (strategy/scenario)
list>
score ind <ind name> <row function>#<list2>
score fun <pre-function expression>
score end (needed to force a recalculate)
```

The number of cell expressions must match the number of columns and there must be at least one indicator if there is a column specification. The `score fun` command is optional. The `score end` command is used to signify the end of the scoreboard rules and to allow the cells to be evaluated. This avoids evaluating each cell each time that an indicator is added. *List 2* as referenced above is a hash separated list of cell expressions.

4.4.13 TS2 window options

```
new ts2
```

Opens a new Time Series 2 window. Commands to specify location, size and title, etc, are identical to previous windows. Window specific options are a list of commands of the form `dataset <dataset reference>`. Data sets must have a consistent number of columns and types.

4.4.14 XY window options

```
new xy
```

Opens a new XY window. Options are similar to the Time Series. Each data set is required to mutate to an SSXY type.

4.4.15 Histogram window options

Opens a new Histogram window.

```
new hist
```

Options available in addition to the standard ones include:

```
dataset <dataset reference>
```

sets the data set to use in this window.

```
indices <x> <y> <z> <3d> <bins>
```

where *x*, *y*, *z* are the columns (0 based) to use, *3d* is a `0` if the graph is 2D or `1` for a 3D version. The window will use defaults if the *z* index is invalid. The *bins* value indicates the number of bins to use:

```
rule <num> <rule>
```

where num is 0, 1, 2 or 3 and represents the 2D, 3D, minimum and maximum rule respectively.

```
type <value>
```

where *value* is either 0, 1 or 2 representing a number, time or string based X dimension respectively.

4.5 Parser/rule specifics

The parser is a simple recursive decent parser although the expressions usually require some soul searching and a good knowledge of the data to be processed in order to decipher the meaning. The parser operates on *tokens* and all tokens have both a type and a value.

4.5.1 Token types

Tokens can be broadly separated into three groups, the first being scalar, the second being an array and the third being a vector of arrays. A vector in this instance is a dynamic 1D vector containing a fixed number (possibly of differing lengths) of arrays rather than the classical physical direction and length version.

The tokens themselves have specific types and include the common programming types of *Number* (double precision), *String*, and *Boolean* although there are generally multiple versions of each corresponding to each of the three groups. To be useful in a spatial and graphical environment, the token types have been extended to include *Point*, *Colour*, *Dataset*, *Polygon*, *Line* and *Time*. Currently the *Polygon* and *Line* types are only available as a scalar type as generally a *Data set* is better at encompassing their structure.

4.5.2 Parser operations

The parser operates on *tokens* in a similar manner to a matrix based language. For example a scalar added to an array of numbers returns an array of numbers. A scalar added to a vector of arrays of numbers returns the vector with the scalar being added to each element in each of the arrays. When it comes to adding an array to a vector then each element in the array is added to the corresponding array from the vector. The first element in the array is added to the first array from the vector, the second element is added to the second array in the vector etc. The result is a vector of arrays.

The dimensions are critical and must match. For array to vector functions to work the size of the array must match the number of arrays in the vector. From a simple perspective, functions which reduce the amount of data will resolve the data from a vector to an array to a scalar in that order. For example the minimum of a vector will return an array with the array dimensions equalling the number of arrays in the vector. The elements in this array will correspond to the minimum of each of the arrays. The minimum of this array will then return the scalar result. Functions which increase the amount of data generally do so by combining multiple tokens from one group into a single token of a higher group. An example is the *strlist()* function which creates a string array from individual strings.

4.5.3 Operators

The parser includes common mathematical and boolean operators. Standard mathematical operators are `* / % + -`. The standard boolean operators are `< > <= >= == != ! && ||`.

4.5.4 Expression building

Although the hierarchy is complex it simplifies the handling of multiple data sets. The majority of expressions the parser processes involves dealing with attributes of data sets and the processing generally follows the trend described below.

Data sets which are of interest and in most instances correspond to a particular strategy or scenario are initially selected. Functions are available which take one or multiple keywords and search available data sets, returning an array of data sets, which match the desired pattern. The keywords are usually specific to a strategy or scenario name which is currently extracted from part of the data set path hierarchy. This is stored as part of the metadata associated with a data set and may be changed if desired.

Expressions are then used to extract those attributes from the data sets which are of interest. This is analogous to extracting a column from a table, however when dealing with multiple data sets the returned type will be a vector of arrays of the type of the attribute requested. Complex expressions often require multiple attributes from each data set so there are usually multiple attribute queries for the same data sets.

Finally the attributes are summarised. Averaging, summing or finding percentiles are a common usage as is determining a colour based on predetermined limits or targets.

4.5.5 Expression syntax

The expression syntax is simple infix notation. The standard mathematical and boolean, binary and unary operators are available and work on the number types including arrays and vectors. Parentheses are used for priority as is usual for infix expressions.

The parser recognises functions (see section 4.6) and has the usual *function_name(arg1,...)* syntax. Most of the mathematical functions *abs, max, min, log,...* are implemented, however not all currently operate on the complete range of types. The vector types were added after the initial implementation and so not all of the functions correctly handle vector processing.

Some functions such as *head()* and *tail()* work across multiple types. Other functions are more pedantic in their argument types. Some will try to convert the argument type into something which is usable, for example passing a string to a function which expects a data set. In this case some functions will attempt to resolve the string to the data set requested although caution is needed as some queries will return multiple data sets
(i.e. data set array) which may not be the desired result.

Strings do not need to be double quoted however it is recommended as the inclusion of non alphanumeric characters may cause ambiguities. A hyphen character, for example in the middle of a string, will attempt a string subtraction which will fail.

Some functions have multiple implementations depending upon the number of arguments that are passed. For example *min()* works as expected when passed a single array, however when passed two arrays an array is returned which is the pair wise minimum of the elements of the arrays.

4.5.6 Multiple expressions

An expression can consist of multiple expressions separated by a semicolon. Assignments are done to variables which begin with a $ character. If followed by an underscore (e.g. $_ds) the variable is temporary and only visible during the evaluation of the current line and is removed after parsing. This is commonly used to allow intermediate variables in an expression whilse avoiding huge numbers of global variables and the corresponding variable name space issues. Assignments may also be made to data sets, however currently the only implemented behaviour is with the colour attribute. The general form of an assignment is
`datasetname.attribute = value`

4.5.7 Static variables

There are some *static* variables for colours, which are predefined. These include $colour.red, $colour.blue,... An expression such as
`ptfile.colour=$colour.red` will colour all points, lines or polygons in the *ptfile* dataset red.

4.5.8 Symbol resolution

Outside of a function the parser will attempt to resolve strings and convert them into a data set or array of data sets. If unsuccessful the string will be left as is. This has a habit of doing the unexpected and it is recommended that all queries to resolve strings to data sets be done implicitly using the data set lookup functions.

4.6 Functions

Mathematical functions currently (or partially) implemented include min, max, avg, abs, exp, pow, log, sqrt, sin, cos, tan, asin, acos, atan, ceil and floor.

4.6.1 min

min() calculates the minimum value from either scalar or arrays of numbers.

```
number min( number n ) [degenerative]
number min( numarray n )
numarray min( vnumarray n )
number min( number n1, number n2 )
numarray min( numarray n1, number n2 )
numarray min( number n1, numarray n2 )
numarray min( numarray n1, numarray n2 )
```
Returns the minimum value.

4.6.2 max

max() calculates the maximum value from either scalar or arrays of numbers.

```
number max( number n ) [degenerative]
number max( numarray n )
numarray max( vnumarray n )
number max( number n1, number n2 )
numarray max( numarray n1, number n2 )
numarray max( number n1, numarray n2 )
numarray max( numarray n1, numarray n2 )
```

Returns the maximum value.

4.6.3 avg

avg() calculates the average value from a set of numbers. If multiple arguments are supplied all arguments must be of the same type and size. In the latter case where the arguments are number arrays the average values returned are the average of each index from each of the arrays.

```
number avg( numarray n )
number avg( number n,... )
numarray avg( numarray n,... )
```

Returns the average value.

4.6.4 abs

abs() calculates the absolute value from the supplied numbers.

```
number abs( number n )
numarray abs( numarray n )
```

Returns the absolute value.

4.6.5 exp

exp() calculates the natural logarithm raised to the supplied argument.

```
number abs( number x )
numarray abs( numarray x )
```

Returns e^x.

4.6.6 log

log() calculates the natural logarithm of supplied argument.

```
number log( number x )
numarray log( numarray x )
```

Returns ln$x$.

4.6.7 sqrt

sqrt() calculates the square root of the supplied argument.

```
number sqrt( number x )
numarray sqrt( numarray x )
```
Returns $\sqrt{x}$.

4.6.8 sin

sin() calculates the sine of the supplied argument.

```
number sin( number x )
numarray sin( numarray x )
```
Returns sin*x*.

4.6.9 cos

cos() calculates the cosine of the supplied argument.

```
number cos( number x )
numarray cos( numarray x )
```
Returns cos*x*.

4.6.10 tan

tan() calculates the tangent of the supplied argument.

```
number tan( number x )
numarray tan( numarray x )
```
Returns tan*x*.

4.6.11 asin

asin() calculates the arc sine of the supplied argument.

```
number asin( number x )
numarray asin( numarray x )
```
Returns arcsin*x*.

4.6.12 acos

acos() calculates the arc cosine of the supplied argument.
```
number acos( number x )
numarray acos( numarray x )
```
Returns arccos*x*.

4.6.13 atan

atan() calculates the arc tangent of the supplied argument.

```
number atan( number x )
numarray atan( numarray x )
```
Returns arctan*x*.

4.6.14 atan2 [unimplemented]

atan2() calculates the arc tangent of the supplied argument. Unlike atan() it takes two arguments which allows calculation of the correct quadrant.

```
number atan2( number x, number y )
numarray atan2( numarray x, numarray y )
```
Returns arctan*y*/*x*.

4.6.15 ceil

ceil() calculates the integer ceiling of the supplied argument.

```
number ceil( number x )
numarray ceil( numarray x )
```
Returns *ceil*(*x*).

4.6.16 floor

floor() calculates the integer floor of the supplied argument.

```
number floor( number x )
numarray floor( numarray x )
```
Returns *floor*(*x*).

4.6.17 length

length() calculates the length of the supplied argument. For implemented vectors each element of the returned array corresponds to the length of the corresponding array in the vector.

```
1 length( number x )
1 length( string x )
1 length( boolean x )
1 length( point x )
1 length( colour x )
number length( numarray x )
number length( stringarray x )
number length( boolarray x )
number length( pointarray x )
numarray length( vnumarray x )
numarray length( vstringarray x )
```
Returns the length.

4.6.18 tail

tail() calculates the tail of an array or vector. As is the case with similar functions it will resolve a vector to an array by selecting the appropriate element from each array in the vector and combining them into an array.

```
number tail( number x )
string tail( string x )
boolean tail( boolean x )
point tail( point x )
colour tail( colour x )
number tail( numarray x )
string tail( stringarray x )
boolean tail( boolarray x )
point tail( pointarray x )
numarray tail( vnumarray x )
stringarray tail( vstringarray x )
boolarray tail( vboolarray x )
pointarray tail( vpointarray x )
colourarray tail( vcolourarray x )
```

Returns the tail of the array.

4.6.19 head

head() calculates the head of an array or vector.

```
number head( number x )
string head( string x )
boolean head( boolean x )
point head( point x )
colour head( colour x )
number head( numarray x )
string head( stringarray x )
boolean head( boolarray x )
point head( pointarray x )
numarray head( vnumarray x )
stringarray head( vstringarray x )
boolarray head( vboolarray x )
pointarray head( vpointarray x )
colourarray head( vcolourarray x )
```

Returns the head of the array.

4.6.20 in

in() calculates the points which are contained within a polygon data set. No longer commonly used but an example is

```
in(ds(oil_wells),ds("pilbara-trawl"))
```

which returns a boolean array which indicates which *oil_wells* points are contained within the *pilbara-trawl* polygons.

4.6.21 polyattr

```
polyattr( <pointdataset>, <polyds> <attribute_in_polyds>
<null_value> )
```

Checks each point in the pointdata set and finds which polygon in polyds it is inside and returns the associated attribute array for that polygon. If the point is outside the polygon the null_value is returned for that array element. e.g.

```
polyattr($chd_pnts,ds("pilbara-zones"),name,"NULL")
```

This extracts the name attribute from the *pilbara-zones* polygon data set for each point in the *$chd_pnts* point array.

4.6.22 strlist

strlist(string list) converts the scalar strings in the argument list to a string array which is commonly used by the data set lookup functions.

4.6.23 strategy

strategy() takes a single or list of model run names along with a string type (typically an agent name) and returns all data sets (dsarray) which match.

4.6.24 sstrategy

sstrategy() As above except that the string is tested as a substring match instead of from the beginning.

4.6.25 attr

attr() extracts the requested attributes from the supplied data sets. The type returned will correspond to the type of attribute requested. If a data set array is supplied the attribute type which corresponds to the name must match for all data sets.

```
array attr( string str, String attrname)
array attr( dataset ds, String attrname )
vector attr( dsarray dsa, String attrname )
```

Returns the array or vector which contains the data of the attribute(s) requested.

4.6.26 dsets

dsets() takes a list of data sets as its arguments and returns an array of data sets. This function is not used very often.

```
dsarray dsets( object ds,... )
```

ds may be of type data set, dsarray or string with the latter being resolved to more data sets which are added to the returned array. The supplied string is a substring test as opposed to a full string equality test as used in ds().
Returns the array of data sets.

4.6.27 ds

ds() takes a string and resolves to a data set or data set array

```
dataset ds( string str )
dsarray ds( string str )
```

str is the data set name to resolve.
Returns the data set(s) if found

4.6.28 ds1

ds1() as above except returns the head of the data set array if applicable.
No longer implemented use `head(ds(...))` instead.

4.6.29 sort

sort() takes number arrays and returns the sorted array

```
number sort( number n )
numarray sort( numarray n )
```

n is the array to sort
Returns the sorted array
Should also work for vnumarray but isn't yet implemented.

4.6.30 percentile

percentile() takes array and percentile level and returns number.

```
number percentile( number n, number p )[degenerative]
number percentile( numarray n, number p )
numarray percentile( vnumarray n, number p )
```

n is the data array
p is the percentile between 0 and 100 to return
Returns the requested percentile of the supplied data.

4.6.31 value

value() an index array lookup which takes the array and the index and returns the element. This can be considered similar to array indexing using brackets in programming languages such as C. For the vector case each element of the supplied index array is applied to each array in the data vector.

```
number value( numarray data, number index )
string value( strarray data, number index )
boolean value( boolarray data, number index )
point value( pointarray data, number index )
colour value( colarray data, number index )
dataset value( dsarray data, number index )
numarray value( vnumarray data, numarray index )
strarray value( vstrarray data, numarray index )
boolarray value( vboolarray data, numarray index )
pointarray value( vpointarray data, numarray index )
colarray value( vcolarray data, numarray index )
dsarray value( vdsarray data, numarray index )
```

data is the array or vector to derefence.
index is the index position of the array or vector to return (0 based).

4.6.32 vvalue

vvalue() an index lookup for vectors, similar to value() but which returns the array at the given index. This function has a side effect in that it returns a reference to the original data so modification of the returned array will result in modification to the supplied vector.

```
numarray vvalue( vnumarray v, number index )
```

v is the vector to dereference.
index is the index of the array in the vector to return (0 based).
Returns the indexed number array.

4.6.33 tindex

tindex() takes time array and time value and returns the closest index. For the vector case each array is calculated individually. The time arrays supplied must be chronological.

```
number tindex( timearray array, time t )
numarray tindex( vtimearray array, time t )
```

array the time array data.
t the time for which to find the corresponding index.
Returns the index which best represents the temporal location of *t* within the supplied array. If *t* predates the first element of the array the first index (0) is returned. If *t* exceeds the last element of the array the last index is returned. The index returned in other cases will be the last index which does not exceed *t*.

4.6.34 cumsum

cumsum() cumulative sum of numbers. For the vector case each array in the vector is summed individually.

```
number cumsum( number n )[degenerative]
numarray cumsum( numarray n )
vnumarray cumsum( vnumarray n )
```

n is the array to be summed.
Returns the cumulative sum of the supplied data.

4.6.35 limit

limit() returns a limit type with specified thresholds. This allows specification of the two main types of limits used in the front end to allow colourisation or categorisation of numerical data. In both cases there are three states which can be thought of as good, marginal/undecided and bad with colours of green, yellow and red respectively. The thresholds must be consecutive.

```
limit limit( number t1, number t2, number t3, number t4,
boolean inside)
```

t1 is the first threshold value.
t2 is the second threshold value.
t3 is the third threshold value.
t4 is the fourth threshold value.
inside specifies an inner or outer limit type.
Returns the requested limit type.
For an inner limit exceeding either end of the thresholds (or markers) is considered bad. A marginal state is between the outer and inner markers whereas a good state is between both inner markers. An outer limit is the reciprocal, i.e. for an inner limit defined as having thresholds of t1, t2, t3, t4
(x<$t1$)||(x>$t4$) $\Rightarrow$ bad
(x<$t2$)||(x>$t3$) $\Rightarrow$ marginal/undecided
(x>=$t2$)&&(x<=$t3$) $\Rightarrow$ good
and for an outer limit
(x<=$t1$)||(x>=$t4$) $\Rightarrow$ good
(x<=$t2$)||(x>=$t3$) $\Rightarrow$ marginal/undecided
(x>$t2$)&&(x<$t3$) $\Rightarrow$ bad

4.6.36 climit

climit() takes number and limit type and returns the appropriate colour.

```
colour climit( number n, limit l )
colarray climit( numarray n, limit l )
vcolarray climit( vnumarray n, limit l )
```

n is the number(s) to be tested.
l is the limits against which *n* is tested.
Returns a colour(s) based on the given value and limits.

4.6.37 cpanel

cpanel() takes a string, number, or bool plus a colour and returns a cpanel type which is used by the scoreboard to display the value on a given background colour. All values are converted to a string for display.

```
cpanel cpanel( number v, colour c )
cpanel cpanel( string v, colour c )
cpanel cpanel( boolean v, colour c )
```

v is the value to be displayed.
c is the background colour to use for display.
Returns the requested cpanel object.

4.6.38 ternary

ternary() takes a boolean or boolean array plus the true and false arrays or scalars and returns the corresponding array or scalar. In the case of a scalar being supplied as an argument with a boolean array the scalar is first converted into a dimensionally compatible array with each element being equal to the supplied scalar. Where the boolean value supplied is an array the return value is tested element by element for the given arrays.

```
object ternary( boolean b, object t, object f )
colarray ternary( boolarray b, colour t, colour f )
colarray ternary( boolarray b, colarray t, colour f )
colarray ternary( boolarray b, colour t, colarray f )
colarray ternary( boolarray b, colarray t, colarray f )
numarray ternary( boolarray b, numarray t, numarray f )
```

b is the boolean selection.
t is the value returned if *b* is true.
f is the value returned if *b* is false.
Returns the corresponding elements from each of the true and false array or object.

4.6.39 point

point() takes numerical x and y value(s) and converts them to a point based type. For arrays and vectors the dimensions must match.

```
point point( number x, number y )
pointarray point( numarray x, numarray y )
vpointarray point( vnumarray x, vnumarray y )
```

x is the data in the x dimension.
y is the data in the y dimension.
Returns the point(s).

4.6.40 tinline

tinline() slices a surface at the specified x location. The x, y and z array dimensions must match.

```
vnumarray tinline( numarray x, numarray y, numarray z,
number xval )
```

x is the data in the x dimension.
y is the data in the y dimension.
z is the data in the z dimension.
xval is the x value at which to slice the surface.
Returns a vector containing two number arrays, the first the y values and the second contains the z values. This may change at a later date to return a point even though it's not strictly a 2D point in x and y. This needs modifying for use with time series data in the x dimension which may involve modifying the return type.

4.6.41 histbin

histbin() takes x,y arrays, the number of bins and the bin type and generates a binned distribution suitable for display within a histogram window.

```
numarray histbin( numarray x, numarray y, number bins,
string func )
numarray histbin( numarray x, numarray y, number bins,
string func, number percent )
numarray histbin( timearray x, numarray y, number bins,
string func )
numarray histbin( timearray x, numarray y, number bins,
string func, number percent )
```

x is the data in the x dimension.
y is the data in the y dimension.
bins is the number of bins in the output histogram.
func is the function to calculate the binned value. This may currently be *sum*, *avg* (or *average*), *min*, *max*, *percent*, *count* or *med* (or *median*).
percent is the percentile to use which is required with the *percent* function.
Returns data which is suitable for display in the histogram window. This is a number array which contains the y value for each of the bins.

4.6.42 strhistbin

strhistbin() takes x,y arrays and the bin type and generates a binned distribution suitable for display within a histogram window.

```
numarray strhistbin( strarray x, numarray y, string func )
numarray histbin( strarray x, numarray y, string func,
number percent )
```

x is the data in the x dimension.
y is the data in the y dimension.
func is the function to calculate the binned value. This may currently be *sum*, *avg* (or *average*), *min*, *max*, *percent*, *count* or *med* (or *median*).
percent is the percentile to use which is required with the *percent* function.
Returns data which is suitable for display in the histogram window. This is a number array which contains the y value for each of the bins.

4.6.43 histcentres

histcentres() takes an x array and the number of bins and generates the approximate centre values for the binned distribution based on the specified number of bins. This function is used internally and need not be specified but is available for use outside of the histogram window.

```
numarray histcentres( numarray x, number bins )
timearray histcentres( timearray x, number bins )
```

x is the data in the x dimension.
bins is the number of bins.
Returns data which is suitable for display in the histogram window. This is a number or time array with contains the approximate x value for each of the bins.

4.6.44 subset

subset() subsets the given data based on supplied boolean values.

```
numarray subset( boolarray b, numarray data )
timearray subset( boolarray b, timearray data )
colourarray subset( boolarray b, colourarray data )
stringarray subset( boolarray b, stringarray data )
pointarray subset( boolarray b, pointarray data )
vnumarray subset( vboolarray b, vnumarray data )
vtimearray subset( vboolarray b, vtimearray data )
vcolourarray subset( vboolarray b, vcolourarray data )
vstringarray subset( vboolarray b, vstringarray data )
vpointarray subset( vboolarray b, vpointarray data )
```

b contains the boolean values to determine inclusion in the returned object.
data contains the data to subset.
Returns the subsetted data.

4.6.45 cutleft

cutleft() removes the left hand side of a string based on the specified delimiters and count.

```
string cutleft( string str, string delim, number count )
```

string the string to be cut.
delim the delimiters to be used in the cut.
count the number of delimiters to count in the cut.
Returns the cut string.
e.g.

```
cutleft("1234:5678:90",":",1)  ⇒5678:90
cutleft("1234:5678:90",":",2) ⇒ 90
```

4.6.46 cutright

cutright() removes the right hand side of a string based on the specified delimiters and count.

```
string cutright( string str, string delim, number count )
```

string the string to be cut.
delim the delimiters to be used in the cut.
count the number of delimiters to count in the cut.
Returns the cut string.
e.g.

```
cutright("1234:5678:90",":",1) ⇒  1234
cutright("1234:5678:90",":",2)  ⇒1234:5678
```

4.4.47 sum

sum() calculates the sum for the given array. For the degenerative case the input value is returned. For a vector of numbers each array within the vector is summed and an array is returned with each element corresponding to a summed vector.

```
number sum( number data ) [degenerative]
number sum( numarray data )
numarray sum( vnumarray data )
```

data the data to be summed.
Returns the summed data.

4.4.48 makearray

makearray() generates an array from the given scalar objects.

colarray makearray(colour data, ...)
numarray makearray(number data, ...)
booleanarray makearray(boolean data, ...)
strarray makearray(string data, ...)
data an element of the array.
Returns the generated array.

4.4.49 maketable

maketable() generates a table data set from the given arrays.

```
dataset maketable( string name, strarray attrnames, array
data, ... )
```

name is the name of the created data set.
attrnames contains the names of the attributes.
data an attribute of the table.
Returns the generated table data set.

4.4.50 rsum

rsum() sums the elements of a vector of number arrays across the vectors and returns the array of the summed numbers. i.e. the first element from each of the vector arrays is summed and placed into the first element of the returned array. The arrays within the vector must all be the same length. For the degenerative case the input argument is returned.

```
numarray rsum( vnumarray data )
numarray rsum( numarray data ) [degenerative]
number rsum( number data ) [degenerative]
```

data is the vector of number arrays to sum.
Returns the summed array.

4.4.51 tsum

tsum() flattens a vector of time arrays into a single dimension time array and uses this information to sum a vector of numbers for each time record. For the degenerative case the input argument is returned.

```
numarray tsum( vtimearray times, vnumarray data )
numarray tsum( timearray time, numarray data )
[degenerative]
number tsum( time t, number data ) [degenerative]
```

times is the vector of time arrays to flatten.
data is the vector of number arrays to sum.

4.4.52 tsumc

tsumc() flattens a vector of time arrays into a single dimension time array sorted by time and returns the count of the number of records for each time entry. Typically used with *tsumt* and/or *tsum* to calculate averages.

```
numarray tsumc( vtimearray arrays )
```

arrays is the vector of arrays to flatten.
Returns the count array.

4.4.53 tsumt

tsumt() flattens a vector of time arrays into a single dimension time array sorted by time and with duplicate time entries removed. Typically used in conjunction with *tsum* to generate a single time series from multiple data sets.

```
timearray tsumt( vtimearray arrays )
```

arrays is the vector of arrays to flatten.
Returns the flattened time array.

4.6.54 vflat

Not implemented.

4.6.55 uniq

uniq() calculate the unique values of the specified array.

```
strarray uniq( strarray data )
```

data is the array of strings.
Returns the unique values from the data specified.
e.g. `uniq("a,b,c,a")  a,b,c`

4.6.56 aappend

Not implemented.

4.6.57 attach

attach() attaches a column to the given data set. Please note that not all data set types support attaching and those that do usually limit the array types to numbers, times or strings.

```
void attach( dsarray ds, vnumarray arrays, string str )
void attach( dsarray ds, vstrarray arrays, string str )
void attach( dsarray ds, vtimearray arrays, string str )
void attach( dsarray ds, vnumarray arrays, strarray str )
void attach( dsarray ds, vstrarray arrays, strarray str )
void attach( dsarray ds, vtimearray arrays, strarray str )
void attach( dataset ds, numarray array, string str )
void attach( dataset ds, strarray array, string str )
void attach( dataset ds, timearray array, string str )
```

ds is the data set(s) to attach the data to.
arrays is the list of arrays which is attached with each one corresponding to a particular data set within the given dsarray.
array is the array to attach to the given data set.
str is the name(s) of the attribute/column which is attached.

4.6.58 strdup

strdup() converts a string to a string array by duplicating the specified string.

```
strarray strdup( string str, number num )
```

str is the string to duplicate.
num is the number of times to duplicate the string.
Returns the string array.

4.6.59 polygrid

polygrid() converts a polygon data set to a grid/image data set. In the instance where no output grid dimensions other than cell size are specified the output grid will encompass the entire polygon data set dimensions.

```
dataset polygrid( string name, dataset ds, string attr,
number xcell, number ycell, number nullvalue )
dataset polygrid( stringname , dataset ds, string attr,
number xcell, number ycell, number nullvalue, number xmin,
number ymin, number cols, number rows )
```

name is the name of the data set to create.
ds is the polygon data set to convert to a grid.
attr is the name of the attribute column in the polygon data set to use as the grid cell values. The type must be a numarray.
xcell is the size of the x dimension output grid cell.
ycell is the size of the y dimension output grid cell.
nullvalue is the value of the output grid for those locations which are not contained within a polygon.
xmin is the lower left x value of the output grid.
ymin is the lower left y value of the output grid.
cols are the number of columns in the output grid.
rows are the number of rows in the output grid.
Returns the output data set as an image.

4.6.60 scale

scale() scales the given numbers to the specified range.

```
numarray scale( numarray v , number min, number max )
vnumarray scale( vnumarray v , number min, number max )
```

v is the values to scale.
min is the minimum output value.
max is the maximum output value.
Returns the given array(s) scaled to values between the specified minimum and maximum.

4.6.61 num2time

num2time() converts the given number of seconds since 1 January 1970 to a time type.

```
time num2time( number v )
timearray num2time( numarray v )
vtimearray num2time( vnumarray v )
```

v is the number to convert to a time.
Returns the time corresponding to the given number of seconds.

4.6.62 time2num

time2num() converts the given time(s) to a number in unix seconds (seconds since 1 January 1970 00:00:00 UTC). Be aware that some truncation/rounding may occur for certain times due to precision differences between data types.

```
number time2num( time t )
numarray time2num( timearray t )
vnumarray time2num( vtimearray t )
```
t is the time data to convert.
Returns the number(s) corresponding to the given time(s).

4.6.63 surface

surface() generates a surface (tin) from the given points. It is generally considered inefficient and slow and the use of *tinline* is instead recommended.

```
vnumarray surface( numarray x, numarray y, numarray z )
```
x is the x data array of the points.
y is the y data array of the points.
z is the z data array of the points.
Returns a tin data structure (vnumarray type) useable with the *tinslice* routines.

4.6.64 tin

tin() generates a triangular irregular network data array from the given line segment arrays.

```
vnumarray tin( numarray x1, numarray y1, numarray z1,
numarray x2, numarray y2, numarray z2 )
```
x1 is the array of x values corresponding to the initial points of the segments in the surface.
y1 is the array of y values corresponding to the initial points of the segments in the surface.
z1 is the array of z values corresponding to the initial points of the segments in the surface.
x2 is the array of x values corresponding to the last points of the segments in the surface.
y2 is the array of y values corresponding to the last points of the segments in the surface.
z2 is the array of z values corresponding to the last points of the segments in the surface.
Returns a vector containing the tinned data which may be used by the *tinslice* routines. Please note that this function does not create the tin, it only converts the line segments into a format useable by the *tinslice* routines. See *surface* for details on creating a tin from point data or *tinline* for extracting a plane from a surface generated from point data.

4.6.65 tinslicex

tinslicex() extracts a plane from specified tin at the given x value.

```
vnumarray tinslicex( vnumarray tin, number value )
```
tin is the tin data structure previously generated by *tin*.
value is the value at which to extract the plane.
Returns a vector array of the given interpolated y and z values similar to *tinline*. Please use *tinline* instead.

4.6.66 tinslicey

tinslicey() extracts a plane from specified tin at the given y value.

```
vnumarray tinslicey( vnumarray tin, number value )
```
tin is the tin data structure previously generated by *tin*.
value is the value at which to extract the plane.
Returns a vector array of the given interpolated x and z values similar to *tinline*. Please use *tinline* instead.

4.6.67 tinslicez

tinslicez() extracts a plane from specified tin at the given z value.

```
vnumarray tinslicez( vnumarray tin, number value )
```
tin is the tin data structure previously generated by *tin*.
value is the value at which to extract the plane.
Returns a vector array of the given interpolated x and y values similar to *tinline*. Please use *tinline* instead.

4.6.68 tinline

tinline() extracts a plane from a surface.

```
vnumarray tinline( numarray x, numarray y, numarray z,
number value )
```
x is the x data array.
y is the y data array.
z is the z data array.
value is the x value at which to extract the surface.
Returns a vector number array with the first array being the y values and the second the corresponding to the z values for the slice through the surface. The values are interpolated between closest points.

4.6.69 strtime

strtime() converts a string time notation into a time type.

```
time strtime( string value, string format )
```
value is the string representation to convert to time
format is the format of *value*. Currently format may only be `YYYYMMDDHHMMSS`.
Returns the generated time.

4.6.70 icolour

icolour() generates a colour from rgba values.

```
colour icolour( number r, number g, number b, number a )
```
r is the red component of the colour in the range 0 to 255.
g is the green component of the colour in the range 0 to 255.
b is the blue component of the colour in the range 0 to 255.
a is the alpha component of the colour in the range 0 to 255.
Returns the requested colour generated from the component values.

4.6.71 fcolour

fcolour() generates a colour from rgba values.

```
colour fcolour( number r, number g, number b, number a )
```
r is the red component of the colour in the range 0.0 to 1.0.
g is the green component of the colour in the range 0.0 to 1.0.
b is the blue component of the colour in the range 0.0 to 1.0.
a is the alpha component of the colour in the range 0.0 to 1.0.
Returns the requested colour generated from the component values.

4.6.72 carraylimit

carraylimit() generates colour(s) based on given thresholds.

colour carraylimit(number num, colourarray colour_thresholds, numarray number_thresholds)
colourarray carraylimit(numarray num, colourarray colour_thresholds, numarray number_thresholds)
vcolourarray carraylimit(vnumarray num, colourarray colour_thresholds, numarray number_thresholds)
num is the value(s) against which to test the thresholds.
colour_thresholds is an array of colour thresholds which are used for colour selection.
number_thresholds is an array of number thresholds which determine which colour from the colour_thresholds is selected for the given value.
Returns the colour(s) from the given colourarray which correlate to the given thresholds. The length of the colourarray must be one record longer than the numarray to allow for extrapolation. The lower limit of a threshold is tested exclusively with the upper limit being inclusive.

4.6.73 write

write() writes a data set to a file.

```
write( string filename, string filetype, dataset ds )
```
filename is the name of the output file.
filetype is the type of output file. Currently only `pt` and `tbl` files are supported.
ds is the data set to write.
This function does not return anything.

4.6.74 polyarea

polyarea() calculates the areas of the polygons within a polygon data set.

```
numarray polyarea( dataset poly )
```
poly is the polygon data set.
Returns a numarray with each element corresponding to the area of one polygon within the data set.

4.6.75 polyclip

polyclip() clips a polygon data set with a polygon data set.

```
dataset polyclip(string name, dataset subject, dataset poly)
```
name is the data set name for the output data set
subject is the input polygon data set to be clipped
poly is the clipping polygon data set
Returns a polygon data set. Attributes are spatially joined. If both data sets contain attributes with the same name, the polygon attributes are dropped.

4.6.76 mlineclip

Not implemented.

4.6.77 lineclip

lineclip() clips a line data set with a polygon data set.

```
dataset lineclip( string name, dataset line, dataset poly )
```
name is the data set name for the output data set
line is the input line dataset to be clipped
poly is the clip polygon data set
Returns a line data set. Attributes are spatially joined. If both data sets contain attributes with the same name, the polygon attributes are dropped.

4.6.78 pointclip

pointclip() clips a point data set with a polygon data set.

```
dataset pointclip( string name, dataset point, dataset poly )
```

name is the data set name for the output data set
point is the input point data set to be clipped
poly is the clip polygon data set
Returns a point data set. Attributes are spatially joined. If both data sets contain attributes with the same name, the polygon attributes are dropped.

4.6.79 generalise

generalise() uses the Douglas-Peucker algorithm to reduce the number of vertices in both line and polygon data sets.

```
dataset generalise( string name, dataset data, number
tolerance )
```

name is the data set name for the output dataset.
data is the data set to generalise.
tolerance is the tolerance to apply to the generalisation.
Returns the generalised data set.

4.7 Example Rules

In nearly all cases some familiarity with the structure of the data is required. All data sets referenced must be either loaded explicitly through one of the windows or with a preload command during parsing of a Project (NWP) file. In some examples lines have been split for easier reading and reference although each example would normally be on a single line – especially if there are references to temporary (preceding _) variables.

4.7.1 Example of calculating catch trends and displaying using variables

```
$mrlist-strlist(“run1”,”run2”,”run3”);
$ds=sstrategy($_mrlist, fisher)
```

The above sets a temporary variable _mrlist to the list of model runs which will be used as part of the strategy or scenario. Note that although the term strategy or scenario is used the software makes no distinction. Currently the model run names are extracted from the pathname of the data set (the parent directory name of the data set is used) however this may change in future to allow explicit extraction from the data set itself if available. The strlist() function converts its string arguments into a string array which is used by the sstrategy() function to find all loaded data sets for the given list of model runs. The second argument to sstrategy() is the substring within the data set name to match for the selected data sets. This is usually the agent name although care needs to be exercised as something like *fisher* will match *fisher_boats* as well as *fisher_tracks*. The global variable ds is set to match the returned list of data sets which match these parameters.

```
$s2_attr_time=attr($ds, time);
$s2_attr_catch_e=attr($ds, catch_e);
$s2_attr_catch_e=attr($ds, catch_e);
$s2_attr_effort=attr($ds,effort)
```

The above extracts from each data set in the variable ds the requested attribute and places into the mentioned variable. These are all vector arrays with the time being a vector time array the the rest being a vector number array. Each array in the vector corresponds to the attribute array from a data set.

```
$s2_catch_e=tsum($s2_attr_time,$s2_attr_catch_e);
$s2_catch_s=tsum($s2_attr_time,$s2_attr_catch_s);
$s2_effort=tsum($s2_attr_time,$s2_attr_effort);
$s2_time=tsumt($s2_attr_time)
```

The above functions convert the vector arrays into a linear array sorted by time. The time attribute given is required to be in chronological order within each of the arrays. The arrays are then interleaved to form a single chronological array. In the case of tsum() the first argument is the vector of times which is used for sorting yet the second argument is the array (after sorting and interleaving) which is actually returned. A common use of this is where there are multiple boats and each corresponds to an individual data set. The time positions and intervals are not necessarily consistent

between boats, however it is useful to consider them as a combined entity for displaying time against some other attribute. In this instance the choices are Emperor (catch_e), Saurid (catch_s) or Effort. The final line (tsumt()) returns the sorted and interleaved time array in a consistent way such that each number in the previous array corresponds to the correct time.

```
$effort_s2=maketable("effort_s2",strlist("time","effort"),
    $s2_time,$s2_effort );
$catchtrend_e_s2=maketable("catchtrend_e_s2",
    strlist("time","catch_e"),
    $s2_time,$s2_catch_e );
$catchtrend_s_s2=maketable("catchtrend_s_s2",
    strlist("time","catch_s"),
    $s2_time,$s2_catch_s )
```

The above lines create tables from the variables which may be used for plotting in windows using the *Add Data set From Variable* menu options. The first argument to the maketable() function is the name of the table. This is sometimes used for display in the legend and ignored for others (the variable name itself is instead displayed). The second argument is a list of strings which are the column names of the attributes. The final arguments are the attributes to be included in the table. The number of columns in the name list must match the number of attributes supplied.

4.7.2 Example of simple scoreboard rule

The scoreboard uses the same parser as the rulesets and as such has access to the same list of global variables. As each cell is parsed separately it is sometimes useful to include computationally expensive functions with the main rules and simply reference the generated data from a variable within the scoreboard. The main difference between the scoreboard parsing and the main ruleset parsing is that the scoreboard makes available certain variables which are applicable to the cell being parsed (namely *_col* and *_row*). As such most rules are common to the whole row and the cell value is simply a reference to this rule.

```
$_ds=sstrategy($_col,fisher);
$_d=attr($_ds,tot_c_dam);

sum(tail($_d))
```

Calculated total coral damage which is similar in style to the previous example. As coral damage is cumulative we simply need to extract the last value from each of the *fisher* boats using tail() and sum them for all boats. The *_col* variable is a reference to the column heading of the cell being calculated. The rule for each cell in the row is then simply *$_row*.

4.7.3 Example of complex scoreboard rule

```
$_ds=sstrategy($_col,fisher);
$_c=attr($_ds,c_damage);$_b=attr($_ds,b_damage);
$_s=attr($_ds,s_damage);$_sg=attr($_ds,sg_damage);
$_tot=$_c+$_b+$_s+$_sg;
$_pnt=point(attr($_ds,long),attr($_ds,lat));
$_pa=polyattr($_pnt,ds("pilbara-zones"),name,"NULL");
$_data=subset($_pa=="Pilbara_area_1",$_tot);sum(sum($_data))
```

The above calculates the summed damage for Pilbara Area 1. The first three lines are similar to previous examples and extract four *damage* attributes from the *fisher* boats. The fourth line calculates the total damage by summing the four types of damage. There is no claim that this is something which would be reasonable to do and is given only as an example. The fifth line creates a point array from the given x and y positions or in this instance the *long* and *lat* values from the data sets.

The polyattr() function then takes this point array as its first argument and extracts the *name* attribute from the pilbara-zones (note double quotes required due to hyphen) polygon data set for each of the given points. If the point is outside the polygon areas of the data set the string *NULL* is set for that point. Polygons within polygons are ambiguous and the resultant point could be in either polygon. Since most polygon data sets used for this study are non-overlapping (at least for a given time instance) it is worth noting but will generally not be an issue.

The final line subsets the data by selecting those points which have the *Pilbara_area_1* name as their name attribute. The resultant vector array is determined from the _tot variable i.e. we’re extracting the total damage for each point in area 1. The sum() functions are used to initially sum the arrays in the vector and then finally to sum the resultant array to a total damage for all boats within area 1.

4.7.4 Example of simple polygon colouring

```
$jlimit=limit(0,0,0.08,0.4,true);
pt.colour=climit(num(attr(pt,concentration)),$jlimit)
```

This colours the *pt* data set according to the *concentration* attribute and compared against the specified limits. The limit() call sets up the four thresholds and type as previously mentioned for the TS2 window. The attr() call extracts the concentration attribute from the pt file. Since all attributes in a PT file, except *time*, are strings the num() function is required to converted the concentrations to numbers. The climit() call takes this number array and returns a colour array with each colour set according to the limits in the second argument.

4.7.5 Example of simpler polygon colouring

```
pt.colour=ternary(num(attr(pt,concentration))>0.4,
    $colour.red,$colour.green
```

The above tests for the concentration in the PT file being above 0.4 and if so sets the colour to red, otherwise the colour is set to green.

4.7.6 Example of complex polygon colouring (for GeoTS windows)

```
$s2=strlist("run2");
$dlimit=limit(0,0,20000,30000,true);
$chd_ds=sstrategy($s2,fisher);
$_x=attr($chd_ds,long);
$_y=attr($chd_ds,lat);
$chd_pnts=point($_x,$_y);
$chd_pa=polyattr($chd_pnts,ds("pilbara-
zones"),name,"NULL");
$chd_t=attr($chd_ds,time);
$chd_totdamage =
attr($chd_ds,c_damage)+attr($chd_ds,s_damage)+
    attr($chd_ds,sg_damage)+attr($chd_ds,b_damage);
$chd_cumsum=cumsum($chd_totdamage);
$chd_t1=subset($chd_pa=="Pilbara_area_1",$chd_t);
$chd_v1=subset($chd_pa=="Pilbara_area_1",$chd_totdamage);
$chd_c1=cumsum($chd_v1);
$chd_t2=subset($chd_pa=="Pilbara_area_2",$chd_t);
$chd_v2=subset($chd_pa=="Pilbara_area_2",$chd_totdamage);
$chd_c2=cumsum($chd_v2);
$chd_t3=subset($chd_pa=="Pilbara_area_3",$chd_t);
$chd_v3=subset($chd_pa=="Pilbara_area_3",$chd_totdamage);
$chd_c3=cumsum($chd_v3);
$chd_t4=subset($chd_pa=="Pilbara_area_4",$chd_t);
$chd_v4=subset($chd_pa=="Pilbara_area_4",$chd_totdamage);
$chd_c4=cumsum($chd_v4);
$chd_t5=subset($chd_pa=="Pilbara_area_5",$chd_t);
$chd_v5=subset($chd_pa=="Pilbara_area_5",$chd_totdamage);
$chd_c5=cumsum($chd_v5);
$chd_t6=subset($chd_pa=="Pilbara_area_6",$chd_t);
$chd_v6=subset($chd_pa=="Pilbara_area_6",$chd_totdamage);
$chd_c6=cumsum($chd_v6);
```

The above is very similar to a previous example and involves extracting damage attributes for each area. The cumsum() function is used to calculate a cumulative sum for the damage in each area. The limit() is used in the function reference for the geots window which follows.

```
$_val1=sum(value($chd_c1,tindex($chd_t1,$_time)));
$_val2=sum(value($chd_c2,tindex($chd_t2,$_time)));
$_val3=sum(value($chd_c3,tindex($chd_t3,$_time)));
$_val4=sum(value($chd_c4,tindex($chd_t4,$_time)));
$_val5=sum(value($chd_c5,tindex($chd_t5,$_time)));
$_val6=sum(value($chd_c6,tindex($chd_t6,$_time)));
pilbara_z.colour=makearray( climit($_val1,$dlimit),
    climit($_val2,$dlimit),
    climit($_val3,$dlimit),
    climit($_val4,$dlimit),
    climit($_val5,$dlimit),
    climit($_val6,$dlimit));
```

This rule is parsed and executed each time the geots window is drawn. In a similar fashion to the scoreboard the variable *_time* is set to the current value of the time in the window and is available for use within the rule. Each of the first six lines are identical except for the area they reference. The tindex() function extracts the array index of the closest value to the second argument from within the first argument array. The closest value is used as the floor such that the time specified will always be equal to or greater than the value at the index returned. The exception is that if the time of the second argument predates the first time in the array the first index is returned.

The value() function is used in concert with the tindex() function and is in essence an array index dereference. It takes the array in question, the index and returns the value in the array at that index. As initially vectors or arrays are used, the sum() function sums the values across the returned array to give the summed damage for all ’fisher’ boats for that area at the given time. The final lines set the colour of the pilbara_z areas based on the dlimit variable.

The makearray() is used to create an array, the size of which matches the number of polygons in the pilbara_z data set and is in the order of the polygons as previously used. This is not necessarily a clean solution but is currently the only way of combining the different area values into a single array which can be used to assign different colours to each of the areas.

REFERENCES

Esri (1999). Environmental Systems research Institute Inc. *MapObjects® Professional Edition* Redlands, California

Gray, R., E. Fulton, R. Little and R. Scott, (2006). Ecosystem Model Specification within an Agent Based Framework. NWSJEMS Technical Report No. 16.

APPENDIX A: NAMING CONVENTIONS IN NWS-INVITRO AND VISUALISATION SOFTWARE DOCUMENTATION

Symbol	Description/Notes
English alphabet	
A	area
a	age
B	biomass
b	width of an individual (e.g. individual animal)
$\mathbf{C}$	current vector
C	concentration
D	damage (e.g. level of damage done to benthics by cyclones)
e	left this to be the natural number *e*, though usually written exp in the equations; used as a subscript to indicate evader parameters in evader-threat equations
d	depth or distance
dt	time step
F	fishing mortality (numbers taken/killed by fishing operations)
f	stands for feared - used as a subscript to indicate threat parameters in evader-threat equations
g	gut contents
h	stands for hunter - used as a subscript to indicate predator parameters in predator-prey equations
I	Irradiance
i	used as an index in sums etc
j	used as an index in sums etc or as random number placeholder
K	carrying capacity
L	Parameters or values associated with locations or movement (e.g. L_l)
$L(x_t,y_t,z_t)$	location in 3D space at time t
l	length of an individual (e.g. individual animal)
M	numbers dying (mortality)
m	mortality rate
N	numbers (e.g. animal abundance)
n	number of cells in grids etc
p	proportions or probabilities; or stands for prey - used as a subscript to indicate prey parameters in predator-prey equations
q	**catchability – not used yet but left for catchability in FMA agent description**
R	reproduction or recruitment (so numbers spawned etc)
r	Radius
S	spawner biomass
s	Speeds
s_{yr}	seconds in a year
t	time
U	light limitation
$\mathbf{v}$	velocity vectors
$\mathbf{W}$	wind vector
$\mathbf{W_m}$	weight ogive for metabolic functions
w	weight of an individual (e.g. individual animal)
x	longitudinal coordinate (projection in metres)
y	latitudinal coordinate (projection in metres)
z	vertical coordinate

Symbol	Description/Notes
Greek symbols	
α	parameters (e.g. Beverton-Holt alpha)
β	parameters (e.g. Beverton-Holt beta)
γ	parameters (e.g. length of fallow periods)
δ	desirability weightings for animal movement or flags (i.e. anything where calculate a value and if that is > 0 then continue to the next step of the decision tree) or probabilities (where must then draw random number < or > etc to continue on)
ε	parameters
ζ	parameters
η	habitat parameters – suitability, preferences and thresholds
Θ	parameters
θ	parameters and angles
ι	ratings (e.g. of prey or threat – the preferences and fears in the agents files)
κ	rate parameters
λ	coefficients (e.g. taxon specific length-weight parameters)
μ	coefficients (e.g. fecundity limits, growth rates)
υ	parameters
ξ	parameters
π	constant = 3.141….
ρ	proportions
σ	random numbers
τ	parameters
υ	selectivity parameters
Φ	random number
ϕ	diffusion constants
χ	parameters (e.g. number of benthic age classes)
ψ	sediment suitability rating
ω	scalars and weighting factors (e.g. with respect to environmental assessments, and wind and currents when working out velocities)
φ	parameters
ϖ	value of external forcing function
Math symbols	
[]	Use only the integer portion of the value within the [] (i.e. used it where in the code we had a floor() call)

APPENDIX B: PT FILE FORMATS

The format for these files is much like the format for pxm files

```
file:
    PT/pt
    entry
    entry
    ...

entry:
   unsigned char dimension;
   integer ndata;
   char **data;
   integer npt;
   locus *pt;
   integer nvec;
   vector *vec;

locus:
   unsigned char tag;
   real x, y;
```

With the “geometric” info in object comments. Obviously for “point” files
npts is degenerate and will be omitted for the sake of efficiency.
“PT” indicates an ascii format "pt" indicates a raw format.
A “location” format is either in decimal lat/long “l:%g,%g”,
in i-j “i:%d,%d”, or in x-y “x:%.8g,%.8g”.
Odd frame (locus.tag) numbers indicate that the values will be
swapped after reading or before writing. In this instance the tag
describes an aspect of the *externl* format rather than the
in-memory format.

“dimension” is 0, 1, or 2 as the object is a set of
scattered points, a line, or a polygon.

“ndata” is the number of elements in "data" like argc.

“data” is an array of strings like envp or argv and
is itself null if there are no strings at all.

“npt” is the number of points in the object.

“pt” is the data point itself in its own coord frame.

APPENDIX C: PXM FILE FORMAT

PXM/PVM/PFM files:

PXM files are analogous to PBM/PNM files, but based on floating point representations rather than integer. The only changes to the PXM file formats relative to the base PNM formats are as follows:

* the magic numbers P1 to P6 have become F1 to F6. It makes no sense to have floating point bit images, so the library ignores F1 and F2

* the headers are identical apart from inclusion of a "minimum" value before the maximum value

* data is represented as floating point number instead of integers.

For descriptions of PBM and PNM files see
http://netpbm.sourceforge.net/doc/ppm.html
http://netpbm.sourceforge.net/doc/pbm.html

ACKNOWLEDGMENTS

The following people and agencies have contributed significantly to the Study through the provision of technical expertise and advice, and historical data and information. The Study partners gratefully acknowledge their contribution.

Western Australian State agencies
Department of Environment and Conservation (Department of Conservation and Land Management and Department of Environment)
Department of Fisheries
Department of Industry and Resources (Department of Mineral and Petroleum Resources)
Department of Land Information
Department for Planning and Infrastructure (Department of Transport)
Pilbara Tourism Association
Shire of Roebourne
Town of Port Hedland
Tourism Western Australia
Western Australian Land Information System
Western Australian Museum

Commonwealth agencies
Australian Institute of Marine Science
Geoscience Australia (formerly Australian Geological Survey Organisation)

Consultants
Cognito Consulting
David Gordon International Risk Consultants
METOCEAN Engineers (formerly Weather News International, Perth)
Oceanica (formerly DA Lord and Associates)

Industries
Australian Petroleum Production Exploration Association (APPEA)
Apache Energy
BHP Petroleum
Chevron Australia
Dampier Salt
Hamersley Iron
Mermaid Marine
Woodside Energy

Individuals
Clay Bryce
Graham Cobby
Nick D'Adamo
Mike Forde
David Gordon
Andrew Heyward
Barry Hutchins
Bryan Jenkins

Di Jones
Ian LeProvost
Ray Masini
Mike Moran
Steve Newman
Eric Paling
Kelly Pendoley
Bob Prinz
Chris Simpson
Shirley Slack-Smith
Di Walker

Reviewers
Malcolm Haddon

Editorial and publishing
Louise Bell – Graphics/cover design
Lea Crosswell – Webpage design
Rob McKenzie – Editor
Diana Reale – Webpage design
Linda Thomas – Editorial consultant/layout and design
Helen Webb – Editorial consultant/Project Manager

Front cover photos courtesy of:
Centre – Coral reef ecosystem, WA Museum, Clay Bryce
Aquaculture pearls, Department of Fisheries WA
Recreational fishing, Department of Fisheries WA, Jirri Lockman
Offshore petroleum platform, Woodside Energy Ltd
Commercial Fishing, Department of Fisheries WA
Tourism, CSIRO
Coastal development aerial photos, Hamersley Iron Pty Ltd