Contents

Welcome to the new WorldMap Component. This component serves many needs that may arise when an application wants to supplement its interface with a geographical display without the high costs associated with the GPS accurate map systems currently on the market. I believe that this component has a definite place in the geo presentation market. This is the first beta release, and I am hoping to release during Mar 96. Some of the drawbacks associated with the design of this component is the points database that accompanies the component. This component is 100% delphi and depends on no external DLLs or other runtime files other than the datafile. The file WORLDMAP.DAT must be present with the component when it is distributed in an application. Additionally, including the WORLDMAP.DCR in your application allows the ZOOM CURSOR to be used. This use is automatic by the component if it is found within the applications resources.

I have tried to make the help as consistent as possible with DELPHIs help system and have added the appropriate footnotes to enable context sensitive help from the design environment via the F1 key.
I have also added the appropriate jumps into Delphis help for the common Objects.
The help is comprehensive, please use it.

Technical Support

We have attempted to provide the best product possible. Due to the large number of hardware/software configurations represented, the component may not always operate as expected. If you experience any problems with the component we will provide technical assistance to ***registered users*** only for original component and source. We will not provide assistance to developers who have modified the source code. Problems can be reported in the following ways:

- Email: **donbauer@illuminet.net** or **ctech@stanleyassoc.com .**
- FAX 703-683-0039 Attention: Component Technologies
- Compuserve 103730, 1061 Don Bauer. We hope to establish a third party forum for the component, but initially, we will provide support via regular Compuserve mail.

Component Installation

1. Install the TWorldMap component in the Delphi IDE:
   To install the component, select **Options | Install Components** from Delphi's menu. Click the Add button to open the Add Module dialog box. Click the Browse button to open the Add Module file selection dialog box.. Select the full drive and directory name path to the **worldmap.dcu** or **worldmap.pas** file and click the OK button. For example, if you accepted the default installation directory of c:\wmap, you would select c:\wmap\worldmap.dcu.

2. Install the TWorldMap on-line help files:
   Copy the file WMAP.HLP from the \wmap\help directory to your existing \delphi\bin directory. Copy the file WMAP.KWF from the \wmap\help directory to your existing \delphi\help directory. From the Windows Program Manager, run the Delphi Help File Installer program (HelpInst), located in your Delphi Program Group. Click the Open an existing HDX file button or select the File and Open... menu options and open file DELPHI.HDX in your \delphi\help directory. Click the Add a new keyword file button or select the Keywords and Add Keyword File... menu options and add file WMAP.KWF, located in your \delphi\help directory, to the list of keyword files being displayed. Click the Compile and save the current HDX file button or select the File and Save menu options to save and compile the modified HDX file. Depending on your CPU, this process may take up to 30 seconds. Close the Help File Installer window or select the File and Exit menu options to terminate the program.

3. If you plan on using the map component in project directories other than the original install directory, you will need to add the following statement to you Delphi.ini file. With any text editor or the DELPHI IDE, open the file DELPHI.INI in the WINDOWS directory. At the end of the file add the following lines:

```
[TWorldMap]
Mapdir=D:\wmap\data   {or where ever you installed the map data files}
```

Save and Close the DELPHI.INI file. This entry will tell each instance of the map component to look in this directory for the map datafiles. This prevents the need for multiple copies of the map data files throughout the system. If the designated data file resides in the current project directory, it will be used first.

Component Limitations

Within the 16-bit environment, this component is very robust. However, there are some limitations that are inherent to the 16-bit environment, and should be noted when developing with this component.

Maximum number of points stored within the component : 16384

Maximum number of lines stored within the component : 16384

Maximum zoom level : 1 X 2 minutes (1 X 2 miles)

Map Accuracy : The original data is approximately 1:3,000,000,000. The map is very accurate down to levels in the 50-100 mile zoom range and reasonably accurate at lower levels. This component is not GPS accurate and should not be employed where high latitude/longitude resolution is a safety consideration.

Other limitations of the component are specific to certain hardware platforms:

HP Laserjets
A specific hardware problem that has been encountered is the HP Laserjet printers. When printing black and white using either the Laserjet III and Laserjet IV drivers, some printing problems may be encountered. The sollution to this problem is to ensure that the background and land brush colors are set to clWhite and the landbrush style is set to bsClear. This will allow the proper printing of the map.

TWorldMap Component

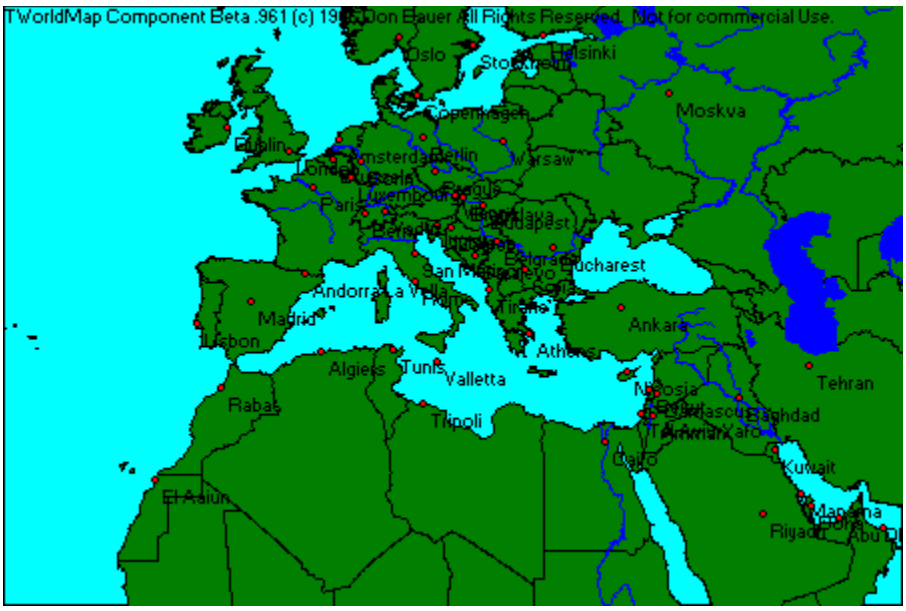
Properties Methods Events Tasks

Unit

WorldMap

Description

The WorldMap Component encapsulates the capabilities of a World Map into the Delphi environment. The component is 100% native delphi and depends on no external DLLs or files other than the map datafile. The map is drawn from a database of points and drawn as polygons or line segments on the client area of the window that it is installed upon. The data file format is detailed in the help system and the component allows for the changing of data files at runtime. In addition to the standard detail levels (Lakes,rivers,countries,states) the user can add up to 27 additional levels of detail and control their display individually at runtime. Here is a ZOOMed example of the Europe from the World Map Demo



Properties

ActiveDraw Property

Applies to
TWorldMap component.

Declaration
```
property  ActiveDraw : boolean;
```

Description
This property was included for developers to turn off the MAP at design time. By Default the WorldMap will draw itself at design time. Due to the nature of the component, this can sometimes
interfere with form design. This property can also be changed at runtime to prevent the Map from
drawing.

Align Property

Applies to
TWorldMap component.

Declaration
```
property  Align: TAlign;
```

Description

The Align property determines how the controls align within their container (or parent control). These are the possible values:

Value	Meaning
alNone	The component remains where you place it in the form. This is the default value.
alTop	The component moves to the top of the form and resizes to fill the width of the form. The height of the component is not affected.
alBottom	The component moves to the bottom of the form and resizes to fill the width of the form. The height of the component is not affected.
alLeft	The component moves to the left side of the form and resizes to fill the height of the form. The width of the component is not affected.
alRight	The component moves to the right side of the form and resizes to fill the height of the form. The width of the component is not affected.
alClient	The component resizes to fill the client area of a form. If a component already occupies part of the client area, the component resizes to fit within the remaining client area.

If the form or a component containing other components is resized, the components realign within the form or control. Using the Align property is useful when you want a control to stay in one position on the form, even if the size of the form changes. For example, you could use a panel component with a various controls on it as a tool palette. By changing Align to alLeft, you guarantee that the tool palette always remains on the left side of the form and always equals the client height of the form.

BorderPen Property

Applies to
TWorldMap component.

Declaration
```
property  BorderPen : TPen;
```

Description
The BorderPen is the pen used to draw the province and state borders. When the doShowStates is TRUE state lines are drawn using this pen.

CurrentLatitude Property

Applies to
TWorldMap component.

Declaration
```
property  CurrentLatitude : integer;
```

Description
The CurrentLatitude property is updated continuously with the cursors current latitude in **minutes** within the current Map view if the EnableUpdatedPoints property is TRUE. A positive number is North, a negative number is South. Allowable range of values is 5400 to -5400. The CurrentLatitude is runtime and read-only.

CurrentLatitudeStr Property

Applies to
TWorldMap component.

Declaration
```
property  CurrentLatitudeStr : string;
```

Description
The CurrentLatitudeStr property is updated continuously with the cursors current latitude within the current Map view if the EnableUpdatedPoints property is TRUE. The resultant value is DD-MM-SS (Degrees, Minutes, Seconds). Allowable range of values is 90-00-00N to 90-00-00S. The CurrentLatitudeStr property is runtime and read-only.

CurrentLongitude Property

Applies to
TWorldMap component.

Declaration
```
property  CurrentLongitude : integer;
```

Description
The CurrentLongitude property is updated continuously with the cursors current longitude in **minutes** within the current Map view if the EnableUpdatedPoints property is TRUE. A positive number is East, a negative number is West. Allowable range of values is 10800 to -10800. The CurrentLongitude property is runtime and read-only.

CurrentLongitudeStr Property

Applies to
TWorldMap component.

Declaration
```
property  CurrentLongitudeStr : string;
```

Description
The CurrentLongitudeStr property is updated continuously with the cursors current longitude within the current Map view if the EnableUpdatedPoints property is TRUE. The resultant value is DDD-MM-SS (Degrees, Minutes, Seconds). Allowable range of values is 180-00-00E to 180-00-00W. Values less than 100 are prefaced with a zero (075-00-00W). The CurrentLongitudeStr property is runtime and read-only.

DetailOptions Property

Applies to
TWorldMap component.

Declaration
```
property DetailOptions : TDetailOptions;
```

Description
The DetailOptions property determines what details are displayed on the map when it is drawn. Since the Map is drawn by a paint message, anytime an option changes, it will appear when the map repaints. Options include:

doShowGrid	When true, this option draws a Latitude/Longitude grid on the map at intervals specified by the GridGapLat and GridGapLong property settings using the Gridpen to draw the lines and the GridFont to display the labels
doShowStates	When true, this option draws the United States state borders with the BorderPen settings
doShowLakes	When true, this option draws lakes with the LakePen, using the fill settings specified by the LakeBrush settings
doShowRivers	When true, this option draws the rivers with the LakePen settings
doShowOwnerFeatures	When true, this option draws owner defined details based upon the current detail settngs. User detail settings are managed by the TurnOnFeature and TurnOffFeature methods.

The different options are completely independant and can be used in any combination, although, rivers and lakes should be shown together (for asthetic purposes only).

EnablePointClick Property

Applies to
TWorldMap component.

Declaration
```
property  EnablePointClick : boolean;
```

Description
The EnablePointClick property sets the components behavior when a point is clicked. If EnablePointClick is TRUE and a point is clicked the OnPointClicked event is triggered. If EnablePointClick is FALSE and a point is clicked the OnPointClicked event will not be triggered. This allows use of the mouse on or around points (as in interactive line drawing) without causing the pointclick event to interrupt the process

EnableUpdatedPoints Property

Applies to
TWorldMap component.

Declaration
```
property  EnableUpdatedPoints : boolean;
```

Description
When the EnableUpdatedPoints is TRUE the CurrentLatitude, CurrentLatitudeStr,CurrentLongitude, CurrentLongitudeStr properties are updated anytime the mouse is moved within the Map area. This allows real-time feedback of the mouse position relative to the latitude and longitude of the cursor on the Map without the developer having to code them.

EnableZoom Property

Applies to
TWorldMap component.

Declaration
```
property  EnableZoom : boolean;
```

Description
The EnableZoom property will enable the Zoom capabilities of the component. After setting the property to TRUE, using the ZoomMouseButton will allow the stretch rectangle to be drawn. When the ZoomMouseButton is released, the enclosed area is zoomed based upon the area bounded by the rectangle and the EnableZoom property is set to FALSE. The EnableZoom property must be set to TRUE before each Zoom operation. When FALSE, holding the ZoomMouseButton down will NOT allow a highlighting an area to be Zoomed.

GridFont Property

Applies to
TWorldMap component.

Declaration
```
property  GridFont : TFont;
```

Description
The GridFont property sets the font used when the grid is drawn. The developer can set color, size, and font type for the labels displayed at the top and left of the screen when the doShowGrids option is set to TRUE.

GridGapLat Property

Applies to
TWorldMap component.

Declaration
```
property GridGapLat : integer;
```

Description
The GridGapLat sets the interval in minutes of the horizontal grid lines. The default value is 600 (10 degrees). This property can be adjusted at runtime to allow finer meshing when zoom levels are increased. Valid values are 1-10800 (although either extreme would be useless). If the valid values are exceeded, the default is used. This property applies when the doShowGrid option is set to TRUE in the DetailOptions property.

GridGapLong Property

Applies to
TWorldMap component.

Declaration
```
property GridGapLong : integer;
```

Description
The GridGapLong sets the interval in minutes of the vertical grid lines. The default value is 900 (15 degrees). This property can be adjusted at runtime to allow finer meshing when zoom levels are increased. Valid values are 1-21600 (although either extreme would be useless). If the valid values are exceeded, the default is used. This property applies when the doShowGrid option is set to TRUE in the DetailOptions property.

GridPen Property

Applies to
TWorldMap component.

Declaration
```
property  GridPen : TPen;
```

Description
The GridPen is the pen used to draw the lines for the Grid. The user has complete control as to how the lines are drawn based upon the pen settings. The grid will show on the map if the doShowGrids option is set to TRUE.

LabelFont Property

Applies to
TWorldMap component.

Declaration
```
property  LabelFont : TFont;
```

Description
The LabelFont property sets the font used for labels associated with points on the map. The developer can set color, size, and font type for the labels displayed when labels are assigned and the ShowPointLabels property is set to TRUE.

LakeBrush Property

Applies to
TWorldMap component.

Declaration
```
property  LakeBrush : TBrush;
```

Description
The LakeBrush sets the way the lakes are filled when drawn. By setting color and patterns, lakes can be drawn to user specifications. The LakeBrush applies when the doShowLakes option is set to TRUE and is used whenever the map is drawn.

LakePen Property

Applies to
TWorldMap component.

Declaration
```
property  LakePen : TPen;
```

Description
The LakePen is the pen used to draw the outlines for the lakes and also to draw the rivers. The user has complete control as to how the lines are drawn based upon the pen settings. The lakes show on the map if the doShowLakes option is set to TRUE and rivers show if the doShowRivers option is set to TRUE.

LandBrush Property

Applies to
TWorldMap component.

Declaration
```
property  LandBrush : TBrush;
```

Description
The LandBrush property of the component sets the color and pattern of the Land Masses when the Map is drawn. The water color is determined by the color of the client area of the form.

LandPen Property

Applies to
TWorldMap component.

Declaration
```
property  LandPen : TPen;
```

Description
The LandPen is the pen used to draw the outlines or more specifically coastlines for the land masses when the map is drawn. The user has complete control as to how the lines are drawn based upon the pen settings. The setting of this pen will determine how the coastline appears on the map.

Mapfile Property

Applies to
TWorldMap component.

Declaration
```
property  Mapfile: string;
```

Description
This property allows the component to draw owner supplied data files. Changing this property from the default WORLDMAP.DAT will force the map to redraw using the new file. This property can be changed at anytime and is available at design time and runtime. All features and capabilites of the component work with the owner file. If the file does not exist an exception is raised.

PointMouseButton Property

Applies to
TWorldMap component.

Declaration
```
property  PointMouseButton : TMouseButton;
```

Description
The PointMouseButton property determines which button activates the OnPointClicked event of the component. The default button is the RIGHT button. If there are points assigned to the map and they are showing, clicking the PointMouseButton
will trigger the OnPointClicked event if assigned.

PointWidth Property

Applies to
TWorldMap component.

Declaration
```
property  PointWidth : integer;
```

Description
This property determines the width in pixels of the points drawn on the map. This applies to the pointstyles psCircle and psSquare. This setting is global and ALL points drawn with the aforementioned pointstyles will draw using this setting.

ShowLines Property

Applies to
TWorldMap component.

Declaration
```
property  ShowLines : boolean;
```

Description
The Showlines property when TRUE will draw any lines stored within the component anytime the component is redrawn. If lines are showing and this property is set to FALSE the map will redraw without the lines.

ShowPoints Property

Applies to
TWorldMap component.

Declaration
```
property  ShowPoints : boolean;
```

Description
The ShowPoints property when TRUE will redraw any points stored within the component anytime the component is redrawn. Changing this property from FALSE to TRUE at runtime will draw any stored points. Changing this property from TRUE to FALSE at runtime will invalidate the map and it will redraw clean (without any points).

ShowPointLabels Property

Applies to
TWorldMap component.

Declaration
```
property  ShowPointLabels : boolean;
```

Description
The ShowPointLabels property when TRUE will draw any labels associated with points stored within the component anytime the component is drawn. Changing this property from FALSE to TRUE at runtime will draw any stored labels. Changing this property from TRUE to FALSE at runtime will invalidate the map and it will redraw the points only (if ShowPoints is TRUE).

StartELongitude Property

Applies to
TWorldMap component.

Declaration
```
property  StartELongitude : longint;
```

Description
The StartELongtitude property holds the Eastern most Longitude in **minutes** (RIGHT) of the current Map view. This property can be set before the Map is initially displayed to show a specific region of the Map, or adjusted at runtime to shift the map view when it is redrawn. This property is updated anytime the scale of the map is changed, as with the ZOOM function. The valid range of values for property are 10800 to -10800, the default value is 10800 (180 degrees * 60 minutes per degree).

Note: Changing this property at runtime will not automatically redraw the map. This is done to allow changing all 4 boundary properties without a redraw between each change.

StartNLatitude Property

Applies to
TWorldMap component.

Declaration
```
property  StartNLatitude : longint;
```

Description
The StartNLatitude property holds the Northern most Latitude in **minutes** (TOP) of the current Map view. This property can be set before the Map is initially displayed to show a specific region of the Map, or adjusted at runtime to shift the map view when it is redrawn. This property is updated anytime the scale of the map is changed, as with the ZOOM function. The valid range of values for this property are 5400 to -5400, the default value is 5400 (90 degrees N * 60 minutes per degree).

Note: Changing this property at runtime will not automatically redraw the map. This is done to allow changing all 4 boundary properties without a redraw between each change.

StartSLatitude Property

Applies to
TWorldMap component.

Declaration
```
property  StartSLatitude : longint;
```

Description
The StartSLatitude property holds the Southern most Latitude in **minutes** (BOTTOM) of the current Map view. This property can be set before the Map is initially displayed to show a specific region of the Map, or adjusted at runtime to shift the map view when it is redrawn. This property is updated anytime the scale of the map is changed, as with the ZOOM function. The valid range of values for this property are 5400 to -5400, the default value is -5400 (90 degrees S * 60 minutes per degree).

Note: Changing this property at runtime will not automatically redraw the map. This is done to allow changing all 4 boundary properties without a redraw between each change.

StartWLongitude Property

Applies to
TWorldMap component.

Declaration
```
property  StartWLongitude : longint;
```

Description
The StartWLongtitude property holds the Western most Longitude in **minutes** (LEFT) of the current Map view. This property can be set before the Map is initially displayed to show a specific region of the Map, or adjusted at runtime to shift the map view when it is redrawn. This property is updated anytime the scale of the map is changed, as with the ZOOM function. The valid range of values for property are 10800 to -10800, the default value is -10800 (180 degrees * 60 minutes per degree).

Note: Changing this property at runtime will not automatically redraw the map. This is done to allow changing all 4 boundary properties without a redraw between each change.

ZoomMouseButton Property

Applies to
TWorldMap component.

Declaration
```
property  ZoomMouseButton : TMouseButton;
```

Description
The ZoomMouseButton property determines which button is activates the zoom feature of the component. Holding the ZoomMouseButton down and dragging creates a rectangle ,that, when the button is released will redraw the map ZOOMing the defined rectangle to the Client area of the form. The ZoomMouseButton can only be used if the EnableZoom property is first set to TRUE.This action updates the StartNLatitude, StartSLatitude, StartWLongitude, StartELongitude properties with the boundaries of the new map area.

Methods

AddLine
AddLineDec
AddLineInt
AddPoint
AddPointDec
AddPointInt
ClearLines
ClearPoints
ConvertLatLongToXY
ConvertLatLongToXYInt
ConvertLatLongToXYDec
ConvertPoint
ConvertXYtoLatLong
DeleteLine
DeletePoint
FindPoint
FixAspectRatio
FullScreen
GetCanvas
GetLine
GetPoint
IsFeatureOn
PrintMap
Plotpoint
PlotPointInt
PlotPointDec
SaveToFile
SetCoordinates
SetCoordinatesInt
SetCoordinatesDec
TurnOffFeature
TurnOnFeature
UpdateLine
UpdatePoint
ZoomInByFactor
ZoomOutByFactor

AddLine Method

Unit
Worldmap

Declaration
```
function AddLine(latfrom, longfrom, latto, longto : string; linecolor :
Tcolor; wide : integer; Atag : longint) : integer;
```

Description
The AddLine method adds a line connecting the latfrom,longfrom point to the latto,longto point in the color and width specified by the linecolor and wide values. Once a line is added to the component, anytime the map redraws and the ShowLines property is TRUE, the lines will be redrawn. The function returns the index value of the line in the LineObject list in the component. The line can be updated by calling GetLine method and the UpdateLine method. The line can be deleted from the list by using the DeleteLine method passing the index value to the method or all lines can be removed by calling the ClearLines method.

AddLineDec Method

Applies to
TWorldMap component.

Declaration
```
function AddLineDec(LatFrom,LongFrom,LatTo,LongTo : double;
linecolor:TColor;wide:integer; Atag : longint):integer;
```

Description
This Method is identical to the AddLine method with one difference: the latitudes and longitudes are passed as **decimal degrees.** This capability was added due to the large number of owner data files that are maintianed in the decimal format. The AddLineDec method draws a line connecting two points based upon Latitude/Longitude using linecolor as the color and wide as the line width in pixels. The line is stored within the component and is redrawn anytime the map is redrawn if the ShowLines property is TRUE. The line algorithm will draw the line accurately if any portion of the line is in the viewport regardless of the end points.

AddLineInt Method

Applies to
TWorldMap component.

Declaration
```
function AddLineInt(iLatFrom,iLongFrom,iLatTo,iLongTo : integer;
linecolor:TColor;wide:integer; Atag:longint):integer;
```

Description
This Method is identical to the AddLine method with one difference: the latitudes and longitudes are passed as integers in **minutes.** The AddLineInt method draws a line connecting two points based upon Latitude/Longitude using linecolor as the color and wide as the line width in pixels. The line is stored within the component and is redrawn anytime the map is redrawn if the ShowLines property is TRUE. The line algorithm will draw the line accurately if any portion of the line is in the viewport regardless of the end points.

AddPoint Method

Applies to
TWorldMap component.

Declaration
```
function Addpoint(sLat,sLong : string; sLabels : TStringList; fFont : Tfont;
    cColor : TColor; Pstyle : TPointStyle; Bitmap : Tbitmap; tag : longint) :
integer;
```

Description
The AddPoint method adds points to the components internal object list. If ShowPoints is TRUE, the points are drawn on the map using the pointstyle and pointwidth. If ShowPointLabels is TRUE, any text in the sLabels stringlist is also displayed first attempting to use the fFont assigned and if fFont is not assigned, the text is drawn using the components Font property. Minimum required (non nil) parameters are sLat,sLong, cColor and Pstyle. Tag, if unassigned defaults to 0. Points can be modified after addition by using the FindPoint, GetPoint and UpdatePoint methods.

Important Note: It is important to send **nil** values in place of unassigned values.

AddPointDec Method

Applies to
TWorldMap component.

Declaration
```
function AddpointDec(Lat,Long : double; sLabels : TStringList; fFont : Tfont;
cColor : TColor; Pstyle : TPointStyle; Bitmap : Tbitmap; tag : longint) :
integer;
```

Description
This Method is identical to the AddPoint method with one difference: the latitude and longitude are passed as integers in **decimal degrees.** The AddPointInt method adds points to the components internal object list. If ShowPoints is TRUE, the points are drawn on the map using the pointstyle and pointwidth. If ShowPointLabels is TRUE, any text in the sLabels stringlist is also displayed first attempting to use the fFont assigned and if fFont is not assigned, the text is drawn using the components LabelFont property. Minimum required (non nil) parameters are Lat, Long, cColor, Pstyle. Tag, if unassigned defaults to 0. Points can be modified after addition by using the FindPoint, GetPoint and UpdatePoint methods.

Important Note: It is important to send **nil** values in place of unassigned values.

AddPointInt Method

Applies to
TWorldMap component.

Declaration
```
function AddpointInt(iLat,iLong : integer; sLabels : TStringList; fFont :
Tfont; cColor : TColor; Pstyle : TPointStyle; Bitmap : Tbitmap; tag : longint)
: integer;
```

Description
This Method is identical to the AddPoint method with one difference: the latitude and longitude are passed as integers in **minutes.** The AddPointInt method adds points to the components internal object list. If ShowPoints is TRUE, the points are drawn on the map using the pointstyle and pointwidth. If ShowPointLabels is TRUE, any text in the sLabels stringlist is also displayed first attempting to use the fFont assigned and if fFont is not assigned, the text is drawn using the components Font property. Minimum required (non nil) parameters are iLat,iLong, cColor and Pstyle. Tag, if unassigned, defaults to 0. Points can be modified after addition by using the FindPoint, GetPoint and UpdatePoint methods.

Important Note: It is important to send **nil** values in place of unassigned values.

ClearLines Method

Unit
Worldmap

Declaration
```
procedure ClearLines;
```

Description
The ClearLines method deletes all lines stored within the component.

ClearPoints Method

Applies to
TWorldMap component.

Declaration
```
procedure ClearPoints;
```

Description
The ClearPoints method deletes all points stored within the component.

ConvertPoint Method

Applies to
TWorldMap component.

Declaration
```
procedure Convertpoint(var dlat, dlong : longint;  lat, long : string);
```

Description
The ConvertPoint method converts a latitude/longitude pair into the respective **seconds** value. Seconds are used internal to the component and provide the highest resolution possible in calculation. To get the minutes from the values divide them by 60. This method is included for completeness.

ConvertXYtoLatLong Method

Applies to
TWorldMap component.

Declaration
```
procedure ConvertXYtoLatLong(var lat, long : longint; X, Y : integer);
```

Description
The ConvertXYtoLatLong method converts an X,Y point pair into the respective Longitude/Latitude **seconds** value. Seconds are used internal to the component and provide the highest resolution possible in calculation. To get the minutes from the values divide them by 60. This method is included for completeness.

ConvertLatLongtoXY Method

Applies to
TWorldMap component.

Declaration
```
function ConvertLatLongtoXY(lat,long:string; var X,Y :integer): boolean;
```

Description
The ConvertLatLongtoXY method will take any latitude, longitude **string** argument and return the screen coordinates X and Y. The acceptable formats for Latitude are DDMMSSO or DD-MM-SSO (DD - Degrees, MM - minutes, SS - seconds, O - orientation N)orth or S)outh). The acceptable formats for Longitude are DDDMMSSO or DDD-MM-SSO (DDD - Degrees, MM - minutes, SS - seconds, O - orientation E)ast or W)est). **It is important to ensure that all unused places are filled with zeros, or an invalid fomat exception will be raised**. If the Latitude/Longitude pair is not currently within the current viewport, the method returns FALSE.If the Latitude/Longitude pair is not currently within the current viewport, the method returns FALSE. This method can be used to allow the developer to set bounds for a circle
or other shape on the canvas based upon the latitude/longitude coordinates used for the Map display.

ConvertLatLongToXYInt Method

Applies to
TWorldMap component.

Declaration
```
function ConvertLatLongtoXYInt(ilat,ilong:integer; var X,Y :integer): boolean;
```

Description
This Method is identical to the ConvertLatLongtoXY method will take any latitude, longitude argument in **minutes** and return the screen coordinates X and Y. The valid range of values for the Latitude is 5400 to -5400, and the valid range of values for the Longitude is 10800 to -10800. If the Latitude/Longitude pair is not currently within the current viewport, the method returns FALSE. This method can be used in conjuction with the OnPaint method to allow the developer to set bounds for a circle or other shape and draw the shape on the canvas relative to the latitude/longitude coordinates used by the Map display.

ConvertLatLongToXYDec

Applies to
TWorldMap component.

Declaration
```
function ConvertLatLongtoXYDec(ilat,ilong:double; var X,Y :integer): boolean;
```

Description
This Method is identical to the ConvertLatLongtoXYInt method will take any latitude, longitude argument in **decimal degrees** and return the screen coordinates X and Y. The valid range of values for the Latitude is 90.0 to -90.0, and the valid range of values for the Longitude is 180.0 to -180.0. If the Latitude/Longitude pair is not currently within the current viewport, the method returns FALSE. This method can be used in conjuction with the OnPaint method to allow the developer to set bounds for a circle or other shape and draw the shape on the canvas relative to the latitude/longitude coordinates used by the Map display.

DeleteLine Method

Unit
Worldmap

Declaration
```
procedure DeleteLine(AIndex:integer);
```

Description
The delete line method removes the line at the index specified from the LineObject list in the component.

DeletePoint Method

Applies to
TWorldMap component.

Declaration
```
procedure DeletePoint(AIndex : integer);
```

Description
The DeletePoint method will delete the point specified by Aindex from the pointlist stored within the component. The next time the map is redrawn the point will not show.

FindPoint Method

Example

Applies to
TWorldMap component.

Declaration
```
function FindPoint(FindWhat : TFindType; searchstring : string; Startndx : integer) : integer;
```

Description
The Findpoint method will search the list of points stored within the component based upon the Find Type and return an index value if the search is successful, or a -1 if the search fails. The search is case sensitive. Possible search fields are Latitude, Longitude in string format, Labels and Tag value. The Startndx value determines which index in the Pointlist to start searching with.

Example

The following example uses a Dialog Box with radio buttons for each find type. When the user clicks the OK button, the code is executed. The search starts with the first object in the list.

```
procedure TFindDlg.OKBtnClick(Sender: TObject);
var
  Index : integer;
begin
  with Main.Worldmap1 do begin
     if RBLat.checked then
       Index := FindPoint(ftLat,searchtext.text,0)
     else if RBLong.checked then
       Index := FindPoint(ftLong,searchtext.text,0)
     else if RBLabel.checked then
       Index := FindPoint(ftLabel,searchtext.text,0)
     else if RBTag.checked then
       Index := FindPoint(ftTag,searchtext.text,0);
     if Index <> -1 then begin
       messagedlg('Success!',mtinformation,[mbok],0);
      end else
       messagedlg('No points found',mtInformation,[mbOk],0);
  end;
end;
```

FixAspectRatio Method

Applies to
TWorldMap component.

Declaration
```
procedure FixAspectRatio;
```

Description
The FixAspectRatio is provided for developers to set their own viewports and ensure that the map draws with the proper perspective. The method ensures a 2:1 ratio between the delta Long and delta
Lat. This method should be called after setting the StartNLatitude, StartSLatitude, StartWLongitude, and StartELongitude properties and before calling Invalidate to redraw the map. This method bulletproofs the setting of the boundary properties.

FullScreen Method

Applies to
TWorldMap component.

Declaration
```
procedure FullScreen;
```

Description
The FullScreen method restores the map to full screen and redraws. To set the map to a specific size and location, use the StartNLatitude, StartWLongitude, StartSLatitude, StartELongitude properties and invalidate the Map. Remember to use FixAspectRatio to set fix the aspect ratio to 2:1 before invalidating the map. Failure to do so could result in a distorted map.

GetCanvas Method

Applies to
TWorldMap component.

Declaration
```
function GetCanvas : TCanvas;
```

Description
The GetCanvas Method returns the components canvas. This gives the developer the freedom to use the map as a backdrop for other operations requiring direct access to the Canvas.

Example

This example uses the GetCanvas method to get the canvas for direct line drawing on the Worldmap. This function allows the interactive drawing of the line on the canvas. First erasing the old line and drawing the new one. If the isfinal value is true we use the R2_COPYPEN to draw it on the canvas.

```
procedure TMapForm.DrawLine(X,Y:integer;IsFinal : boolean);
var
  MyCanvas : TCanvas;
begin
   MyCanvas := Worldmap1.GetCanvas;
   with MyCanvas do begin
      if not isfinal then
        SetROP2(Handle,R2_NOTXORPEN)
      else
        SetROP2(Handle,R2_COPYPEN);
      Pen.Style := pssolid;
      Pen.Color := clRed;
      Pen.width := 1;
      Moveto(FSPt.X,FSPt.Y);
      Lineto(FEPt.X,FEPt.Y);
      Moveto(FSPt.X,FSPt.Y);
      Lineto(X,Y);
      FEPt.x := X;
      FEPt.Y := y;
  end;
end;
```

GetLine Method

Example

Applies to
TWorldMap component.

Declaration
```
procedure GetLine(var MyLine : TLineObject; Aindex : integer);
```

Description
The GetLine method will fill MyLine object the the line object stored at index value Aindex. The calling function must Create the TLineObject before passing the object to the component. If the pointer is dirty but not assigned, the Assigned method will not detect an unassigned pointer and a GPF will result. If the index value does not exist an exception is raised.

Example

This example illustrates the proper use of the Getline and Updateline methods. This procedure get the line at the index value and changes the color to green with a draw. This will cause an immediate update of the line on the screen without causing a map invalidation.

```
procedure TForm1.ChangePoint(Aindex:integer);
var
  MyLine : TLineObject;
begin
  MyLine := TLineObject.Create;
  try
    With worldmap1 do begin
       GetLine(MyLine, Aindex);
       MyLine.lcolor := clGreen;
       UpdateLine(MyLine,Aindex,True);
    finally
      MyLine.Free;
    end;
end;
```

GetPoint Method

Applies to
TWorldMap component.

Declaration
```
procedure GetPoint(MyPoint:TPointObject;Aindex:integer);
```

Description
The GetPoint method will fill MyPoint object the the point stored at index Aindex. The calling
function must Create the object before passing the TPointObject to the component. If the pointer
is dirty but not assigned, the Assigned method will not detect an unassigned pointer and a GPF will
result.

IsFeatureOn Method

Applies to
TWorldMap component.

Declaration
```
function IsFeatureOn(Afeature : integer) : boolean;
```

Description
The IsFeatueOn method returns a boolean value based upon the feature value passed. The user defined features can be turned on with TurnOnFeature and turned off with the TurnOffFeature methods. The valid range of values for this method are 5 to 32. This method only applies to those data files that contain user defined features or details.

PlotPoint Method

Applies to
TWorldMap component.

Declaration
```
procedure Plotpoint(lat, long, labelstr : string; bColor : TColor);
```

Description
The PlotPoint method is a quick way to add a point to the screen. The coordinate values are sent as strings. The point is NOT stored within the component and therefore must be redrawn anytime the map is invalidated. The labelstr can only be a string not a Tstringlist as in the AddPoint method. The bColor determines the color of the circle drawn at the point.

PlotPointInt Method

Applies to
TWorldMap component.

Declaration
```
procedure PlotpointInt(lat, long:longint; labelstr : string; bColor : TColor);
```

Description
The PlotPoint method is a quick way to add a point to the screen. The coordinate values are sent as **minutes**. The point is NOT stored within the component and therefore must be redrawn anytime the map is invalidated. The labelstr can only be a string not a Tstringlist as in the AddPoint method. The bColor determines the color of the circle drawn at the point.

PlotPointDec Method

Applies to
TWorldMap component.

Declaration
```
procedure PlotpointDec(lat, long:double; labelstr : string; bColor : TColor);
```

Description
The PlotPoint method is a quick way to add a point to the screen. The coordinates are sent as **decimal degrees**. The point is NOT stored within the component and therefore must be redrawn anytime the map is invalidated. The labelstr can only be a string not a Tstringlist as in the AddPoint method. The bColor determines the color of the circle drawn at the point.

PrintMap Method

Example

Applies to
TWorldMap component.

Declaration
```
procedure PrintMap(bAddToCurrent:boolean; pxRect:Ptrect; eBkColor:TColor);
```

Description
The PrintMap method will print the map. The method will print the CURRENT VIEW of the map (regardless of the zoom level) including points, labels, lines, lakes, rivers, states, grids and any user detail options showing on the map. The bAddToCurrent parameter allows the map to be printed on a document that has already been started. This allows the map to be included on documents containing other information. The pxRect parameter is a pointer to a rectangle structure containing the destination coordinates on the page to print the map. The coordinates are printer page coordinates, NOT screen coordinates. **It is the developers responsibility to ensure the dimensions of the rect conform to the aspect ratio of the map**. If pxRect is nil, the map prints on the whole page in Landscape mode. The eBkColor parameter specifies which color to use for the background of the map. If printing from a form, pass the forms Color property. To print the map on a black and white printer, set the LandBrush property to bsClear, and pass clWhite for the background color to reduce ink usage and keep the map ledgible.

Example

The following code prints the map on a full page with a white background. Sending pxRect parameter is as nil causes a full screen print in the Landscape mode. All showing features are printed with the current view of the map

```
   Worldmap1.Print(False, nil, clWhite);
```

The following example starts a document, prints a header, then prints two maps on the page at different locations. The pxRect is assigned and sent by reference.

```
procedure TMapForm.Print1Click(Sender: TObject);
var
  Rect: TRect;
  twidth : word;
  offset : word;
begin
  xRect.Top := 100;
  xRect.Left := 100;
  xRect.Right := 1600;
  xRect.Bottom := 1100;
  Printer.Orientation := poPortrait;
  Printer.Title := 'World Map Demo';
  Printer.BeginDoc;
  offset := (printer.pagewidth - 1000) div 2;
  xrect.left := xrect.left + offset;
  xrect.right := xrect.right + offset;
  twidth := LO(GetTextExtent(printer.canvas.handle,'TWorldmap Component
              Print Demo',30));
  Printer.Canvas.TextOut((Printer.pagewidth - twidth)div 2,5,
                         'TWorldmap Component Print Demo');
  Worldmap1.LandBrush.Style := bsClear;
  Worldmap1.PrintMap(True, @xRect, clWhite);
  xRect.Top := 2000;
  xRect.Left := 2000;
  xRect.Right := 3000;
  xRect.Bottom := 2500;
  Worldmap1.PrintMap(True, @xRect, clAqua);
  Printer.Enddoc;
  Worldmap1.LandBrush.Color := clGreen;
  Worldmap1.LandBrush.Style := bsSolid;
end;
```

SaveToFile Method

Applies to
TWorldMap component.

Declaration
```
procedure SaveToFile(strFileName: String; iWidth, iHeight: Integer;eBkColor:
Tcolor);
```

Description
The SaveToFile method creates a bitmap file of the map. The method will put the CURRENT VIEW of the map (regardless of the zoom level) including points, labels, lines, lakes, rivers, states, grids and any user detail options showing into the bitmap. The srtFilename parameter is the name and path of the bitmap file. The iWidth parameter is the bitmap dimension in pixels, the iHeight parameter is the bitmap dimension in pixels. **It is the developers responsibility to ensure the dimensions of the bitmap conform to the aspect ratio of the map**. The map is drawn to the bitmap specifications. The eBkColor parameter determines the background (water) color of the bitmap.

SetCoordinates Method

Applies to
TWorldMap component.

Declaration
```
procedure SetCoordinates(lat,long : string; scale : integer);
```

Description
The SetCoordinates method positions the Map with the *center* located at the latitude and longitude specified with the scale determining the size of the display area. Any format errors raise an exception.

The latitude format is DDMMSSO [DD=Degrees, MM=Minutes,SS=Seconds,O=Orientation as N)orth or S)outh], for example **350029N**. For Latitude the range for Degrees is 0-90, Minutes is 0-59, seconds is 0-59.

The longitude format is DDDMMSSO [DDD=Degrees, MM=Minutes,SS=Seconds,O=Orientation as E)ast or W)est], for example **0751532E**. For Longitude the range for Degrees is 0-180, Minutes is 0-59, seconds is 0-59.

SetCoordinatesInt Method

Applies to
TWorldMap component.

Declaration
```
procedure SetCoordinatesInt(lat,long : longint; scale : integer);
```

Description
The SetCoordinatesInt method positions the Map with the *center* located at the latitude and longitude specified with the scale determining the size of the display area. The coordinate values are sent as **minutes**

SetCoordinatesDec Method

Applies to
TWorldMap component.

Declaration
```
procedure SetCoordinatesDec(lat,long : double; scale : integer);
```

Description
The SetCoordinatesDec method positions the Map with the *center* located at the latitude and longitude specified with the scale determining the size of the display area. The coordinate values are sent as **decimal degrees.**

TurnOnFeature Method

Applies to
TWorldMap component.

Declaration
```
procedure TurnOnFeature(Afeature : integer) : boolean;
```

Description
The TurnOnFeatue method enables the drawing of a user defined feature based upon the feature value passed. The user defined features can be turned off with the TurnOffFeature method. To check a particular feature use the IsFeatureOn method. The valid range of values for this method are 5 to 31. This method only applies to those data files that contain user defined features or details.

TurnOffFeature Method

Applies to
TWorldMap component.

Declaration
```
procedure TurnOffFeature(Afeature : integer; DoWeDraw : boolean);
```

Description
The TurnOffFeatue method disables the drawing of a user defined feature based upon the feature value passed. If the DoWeDraw value is TRUE the map invalidates and redraws without the feature. If it is FALSE the map does not redraw, but when the map repaints the feature will not appear. The user defined features can be turned on with the TurnOnFeature method. To check a particular features current state use the IsFeatureOn method. The valid range of values for this method are 5 to 31. This method only applies to those data files that contain user defined features or details.

UpdateLine Method

Applies to
TWorldMap component.

Declaration
```
procedure UpdateLine(MyLine:TLineObject;Aindex:integer;DoWeDraw : boolean);
```

Description
The UpdateLine method allows Lines stored in the LineList to be updated. A call to GetLine must be executed prior to calling UpdateLine, to get the Line from the component. All fields of the Line object can be updated. If points are moved, the component will recalculate the internal raw lats and longs. If DoWeDraw is TRUE, the Map will invalidate and redraw showing the updated line, if false, the map wil not redraw. This can allow mass updates, with the last call to UpdateLine using DoWeDraw set to TRUE, to prevent constant redrawing of the map. This method must be used with care as changing all or some of the fields may yield unexpected results. I have included it to allow more flexibility from the developers perspective.

UpdatePoint Method

Applies to
TWorldMap component.

Declaration
```
procedure UpdatePoint(MyPoint : TpointObject; Aindex : integer; DoWeRedraw : boolean);
```

Description
The UpdatePoint method allows points stored in the PointList to be updated. A call to GetPointmust be executed prior to calling UpdatePoint, to get the point from the component. All fields of the point object can be updated. Updating myspace will have no effect as it is recalculated whenever the map is zoomed to readjust the rectangle of space that is live. If points are moved, the component will recalculate the internal raw lat and long . There are two ways to update the location of a point:

1. Setting the dlat and dlong values of the Object with the appropriate seconds values **AND** setting the Lat and Long string values to the empty string. (The update method will recalculate the string lat and long values)
2. Set the Lat and Long values to the new position in the proper string format. The acceptable formats for Latitude are DDMMSSO or DD-MM-SSO (DD - Degrees, MM - minutes, SS - seconds, O - orientation N)orth or S)outh). The acceptable formats for Longitude are DDDMMSSO or DDD-MM-SSO (DDD - Degrees, MM - minutes, SS - seconds, O - orientation E)ast or W)est).

If DoWeDraw is TRUE, the Map will invalidate and redraw showing the updated point, if false, the map wil not redraw. This can allow mass updates, with the last call to UpdatePoint using DoWeDraw set to TRUE, to prevent constant redrawing of the map. This method must be used with care as changing all or some of the fields may yield unexpected results. I have included it to allow more flexibility from the developers perspective.

ZoomInByFactor Method

Applies to
TWorldMap component.

Declaration
```
procedure procedure ZoomInByFactor(lat,long : longint; factor : integer);
```

Description
The ZoomInByFactor method allow the map to be zoomed by a fixed amount with the center of the map located at the lat, long specified. The factor determines the extent to which the map is zoomed. Factor must be a multiple of 2, except for a special case of using 1 to effectively pan the map. The factor process is cumulative, that is, calling the method with a zoom factor of 2, and subsequently calling it with a zoom factor of 4 will yield a total zoom factor of 8. Conversly calling the method with a zoom factor of 2, and calling the method again with a zoom factor of 2 will yield a total zoom factor of only 4. If the zoom area is near the edge of the map, the center is adjusted accordingly.

ZoomOutByFactor Method

Applies to
TWorldMap component.

Declaration
```
procedure procedure ZoomOutByFactor(factor : integer);
```

Description
The ZoomOutByFactor method will zoom the map out keeping the current center of the map. If the method is called with a zoom factor that results in an area greater than the world map dimensions, the map will default to full screen.

Scale

The scale property determines the physical dimensions of the map viewing area. In nautical terms a
minute represents 1 mile, since a Dregree is 60 minutes it is also 60 miles. Specifying a scale of 600 miles would yield a display of 10 degrees wide by 5 degrees high. (Remember the 2:1 aspect ratio).

Events

OnClick
OnDblClick
OnLineFilter
OnMapMouseDown
OnMapMouseMove
OnMapMouseUp
OnPaint
OnPointChange
OnPointClicked
OnPointFilter
OnZoom

OnClick Event

Applies to
TWorldMap component.

Declaration
```
property   OnClick: TNotifyEvent;
```

Description

The OnClick event occurs when the user clicks the component. Typically, this is when the user presses and releases the primary mouse button with the mouse pointer over the component. This will only occur when the EnableZoom property is FALSE.

OnDblClick Event

Applies to
TWorldMap component.

Declaration
```
property  OnDblClick: TNotifyEvent;
```

Description
The OnDblClick event occurs when the user double-clicks the mouse button while the mouse pointer is over the component. This will only occur when the EnableZoom property is FALSE.

OnLineFilter Event

Example

Applies to
TWorldMap component.

Declaration
```
property  OnLineFilter : TLineFilterEvent;
```

Description
The OnLineFilter event is called passing each line in the Linelist to the function for evaluation. If the result is TRUE, the line is drawn on the map each time the lines are drawn. Caution should be used when coding the filter functions, since **each** line is passed through it ***every time*** the map is drawn and ShowLines is TRUE.

Example

The follwing example tests each line passed in against the condition of dlatto value < 0. With this event assigned, only lines with a dlatto value less than 0 will be drawn whenever the lines are drawn.

```
function TForm1.WorldMap1LineFilter(Sender: TObject;
  Curline: TLineObject): Boolean;
begin
  result := false;
  If Curline.dlatto < 0 then result := 1;
end;
```

OnMapMouseDown Event

Applies to
TWorldMap component.

Declaration
```
property  OnMapMouseDown : TMapMouseDownEvent;
```

Description
If assigned, the OnMapMouseDown will be triggered whenever the mouse is depressed within the map client area. In addition to the traditional mouse parameters the latitude and longitude in **seconds** are passed with this event. Seconds are used to allow the highest possible accuracy relative to the map.

OnMapMouseMove Event

Applies to
TWorldMap component.

Declaration
```
property  OnMapMouseMove : TMapMouseMoveEvent
```

Description
If assigned, the OnMapMouseMove will be triggered whenever the mouse is moved within the map client area. In addition to the traditional mouse parameters the latitude and longitude in **seconds** are passed with this event. Seconds are used to allow the highest possible accuracy relative to the map.

OnMapMouseUp Event

Applies to
TWorldMap component.

Declaration
```
property  OnMapMouseUp : TMapMouseUpEvent;
```

Description
If assigned, the OnMapMouseUp will be triggered whenever the mouse is released within the map client area. In addition to the traditional mouse parameters the latitude and longitude in **seconds** are passed with this event. Seconds are used to allow the highest possible accuracy relative to the map.

OnPaint Event

Applies to
TWorldMap component.

Declaration

```
property OnPaint: TNotifyEvent;
```

Description
The OnPaint event occurs when Windows requires the form or paint box to paint, such as when the form or paint box receives focus or becomes visible when it wasn't previously. Your application can use this event to draw on the canvas of the form or paint box.

OnPointChange Event

Example

Applies to
TWorldMap component.

Declaration
```
property   OnPointChange: TNotifyEvent;
```

Description
The OnPointChange event occurs whenever the mouse moves within the Map area (Client Area) of the Form. The Event will only be triggered if the EnableUpdatePoints property is TRUE. This event allows handling to be associated with the movement of the mouse relative to the map (e.g. Keeping updated point positions in a panel showing cursor latitude/longitude).

Example

The following example uses the PointChange event to update two edit fields on a panel with the map form. Each time the mouse moves within the component the edit fields are updated showing the mouse cursors relative Lat and Long based upon the map.

```
procedure TMain.WorldMap1PointChange(Sender: TObject);
begin
  Curlat.text := Worldmap1.CurrentLatitudeStr;
  Curlong.text := Worldmap1.CurrentLongitudestr;
end;
```

OnPointClicked Event

Example

Applies to
TWorldMap component.

Declaration
```
property  OnPointClicked : TPointClickEvent;
```

Description
If assigned, the OnPointClicked will be triggered whenever a point on the map is clicked. The hotspot area of the point object is the actual point or the picture displayed. To disable the OnPointClicked event to allow mouse use around points, set the EnablePointClick property to FALSE:

Example

OnPointClicked event is only triggered if a point on the map is clicked and the EnablePointClick property is TRUE. When called the event sends the point that was clicked to the calling unit. The following code shows the 4 strings associated with the clicked point. The #13 is a carriage return.

```
procedure TMapMain.WorldMap1PointClicked(Sender: TObject; Curpoint:
    TPointObject; AIndex: Integer);
begin
   messagedlg('Current Point is '+inttostr(Aindex)+#13+
               Curpoint.labels.strings[0]+#13+
               Curpoint.labels.strings[1]+#13+
               Curpoint.labels.strings[2]+#13+
               Curpoint.labels.strings[3],mtInformation,[mbOk],0);
end;
```

OnPointFilter Event

Example

Applies to
TWorldMap component.

Declaration
```
property  OnPointFilter : TPointFilterEvent;
```

Description
The OnPointFilter event is called passing each point in the pointlist to the function for evaluation. If the result is TRUE, the point is drawn on the map each time the points are drawn. Caution should be used when coding the filter functions, since **each** point is passed through it ***every time*** the map is drawn and ShowPoints is TRUE.

Example

The follwing example tests each point passed in against the condition of tag value > 1000. With this event assigned, only points with a tag value greater than 1000 will be drawn whenever the points are drawn.

```
function TForm1.WorldMap1PointFilter(Sender: TObject;
  Curpoint: TPointObject): Boolean;
begin
  Result := false;
  if Curpoint.tag > 1000 then result := 1;
end;
```

OnZoom Event

Applies to
TWorldMap component.

Declaration
```
property  OnZoom: TNotifyEvent;
```

Description
The OnZoom event specifies which event handler should execute when the component is Zoomed in or out. This event happens after the map has been zoomed or restored to full screen.

TLineFilterEvent Type

Applies to
TWorldMap component.

Declaration

```
 TLineFilterEvent = function(Sender: TObject; Curline : TLineObject):boolean
of object;
```

Description
The TLineFilterEvent is a custom event designed to allow lines to be filtered before they are drawn. The custom event passes the each line object in the linelist to the calling program allowing the developer to create custom filtering code and returns TRUE or FALSE. If the event returns true, the line is drawn each time lines are drawn on the map. This event does not require the BDE or any database connectivity.

TMapMouseMoveEvent Type

Unit
Worldmap

Declaration
```
TMapMouseMoveEvent = procedure(Sender: Tobject; Shift: TShiftState; X, Y:
integer;Lat,Long : longint) of object;
```

Description
The TMapMouseMoveEvent is a custom event designed to allow the latitude and longitude to be passed (in **seconds**) along with the traditional mouse data to the components parent to obtain information relative to the map coordinate system when the mouse is moved.

TMapMouseUpEvent Type

Unit
Worldmap

Declaration
```
TMapMouseUpEvent = procedure(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer;Lat,Long : longint) of object;
```

Description
The TMapMouseUpEvent is a custom event designed to allow the latitude and longitude to be passed (in **seconds**) along with the traditional mouse data to the components parent to obtain information relative to the map coordinate system when the mouse is released.

TMapMouseDownEvent Type

Unit
Worldmap

Declaration
```
TMapMouseDownEvent = procedure(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer;Lat,Long : longint) of object;
```

Description
The TMapMouseDownEvent is a custom event designed to allow the latitude and longitude to be passed (in **seconds**) along with the traditional mouse data to the components parent to obtain information relative to the map coordinate system when the mouse is down.

TPointClickEvent Type

Unit
Worldmap

Declaration
```
TPointClickEvent = procedure(Sender : TObject; Curpoint : TPointObject;
AIndex:integer) of object;
```

Description
The TPointClickEvent type is a custom event designed to allow a point object to be passed to the forms parent to obtain information about a point when the point is clicked.

TPointFilterEvent Type

Applies to
TWorldMap component.

Declaration

```
 TPointFilterEvent = function(Sender: TObject; CurPoint :
TPointObject):boolean of object;
```

Description
The TPointFilterEvent is a custom event designed to allow points to be filtered before they are drawn. The custom event passes each point object in the pointlist to the calling program allowing the developer to create custom filtering code and returns TRUE or FALSE. If the event returns TRUE, the point is drawn each time points are drawn on the map. This event does not require the BDE or any database connectivity.

TPointObject Type

Unit
Worldmap

Declaration

```
TPointObject = class(TObject)
  Labels : TStringList;
  Picture : TBitmap;
  Color : TColor;
  Font : TFont;
  PointType : TPointStyle;
  Lat : string;
  Long : string;
  Dlat : longint;
  Dlong : longint;
  MySpace : Trect;
  Data : Pointer;
  Tag : longint;
public
   Constructor Create;
   Destructor Destroy;
end;
```

Description
The TPointObject is a custom object that holds information related to points stored within the component in a Tlist Object. The points can be added using AddPoint, deleted with DeletePoint, and updated using the GetPoint and UpdatePoint. FindPoint will find a point based upon the FindType and the search string. All list operations are managed by exception handling internal to the component. The data pointer allows user data to be attached to the point object. Data can only be attached via the GetPoint and UpdatePoint Methods. The constructor ensures that all values are properly initialized. Default values for the TPointObject are : dlat, dlong, tag, myspace 0, Labels - nil, Picture - nil, pointtype - psCircle, Font - created, color - clRed, Lat, Long - . The Destructor ensures that all addional allocations are properly freed prior to calling the Objects destroy method.

TLineObject Type

Unit
Worldmap

Declaration
```
TLineObject  = class(TObject)
   dlatfrom : longint;
   dlatto : longint;
   dlongfrom : longint;
   dlongto : longint;
   lcolor : TColor;
   lwidth : integer;
   Data : Pointer;
   tag : longint;
  public
    Constructor Create;
  end;
```

Description
The TLineObject is a custom object used to store line information in the component. The constructor initializes the values in the object to their default values: latfrom, longfrom, latto, longto default to 0, lcolor defaults to clBlue, lwidth defaults to 2, Data defaults to nil, tag defaults to 0.

TFindType

Unit
Worldmap

Declaration
```
TFindType = (ftLabel,ftTag,ftLat,ftLong);
```

Description
FindType determines which part of a TPointObject is searched when the FindPoint method is called.
ftLabel searches the TStringList , ftTag searches based upon a Tag value, ftLat searches based upon the Latitude, ftLong searches based upon the Longitude.

TPointStyle Type

Unit
Worldmap

Declaration
```
TPointStyle = (psCircle,psSquare,psNone,psLineto,psPicture);
```

Description
Pointstyle determines what kind of point is used when points are drawn on the map. Using the UpdatePoint method, pointstyle can be changed based upon changing conditions. Pointstyles other than psPicture and Psnone will use the color stored inside the TPointObject when drawing the point.

TDetailOptions Type

Unit
Worldmap

Declaration
```
TDetailOption = (doShowStates, doShowLakes, doShowRivers, doShowGrid,
doShowOwnerFeatures);
```

Description
The TDetailOptions type is a set of valid options that can be used for map display. The different elements are explained in the DetailOptions property.

ZOOM

The WorldMap has built in ZOOM Capabilities. When the EnableZoom property is set to TRUE and the appropriate ZoomMouseButton is selected, holding the zoom button down and dragging, creates a rectangle on the Map.

When the button is released, the Map redraws with the rectangle as the new area for the map.

Although this capability is built into the map component, the same thing can be accomplished by resetting the StartNLatitude, StartSLatitude, StartWLongitude, StartELongitude properties to the desired display area, and Invalidating the canvas of the Client area containing the map.

Important points

1. Then Map as designed is a Mercator Projection. The system expects a 2:1 aspect ratio between Longitude and Latitude respectively in order to draw the Map in the proper perspective. **Failure to maintain the 2:1 ratio will result in a distorted map!**. When using the built in ZOOM capabilities of the component, the component will always ensure a 2:1 ratio. For those who wish to set their own map bounds, the method FixAspectRatio will update the points to ensure a 2:1 ratio. Calling this method before redrawing the Map will ensure a proper perspective on the Map. Implementing the Zoom feature programatically will allow the developer to set view regions and switch between

them (i.e. Europe, North America, etc). Using the ZOOM feature does not disable or prevent the programmer from programatically setting Map bounds.

Tasks

Creating Owner Data Files
Drawing Directly On the Map Canvas
Hooking Database Events to PointClicked Events
Attaching User Data to Points and Lines

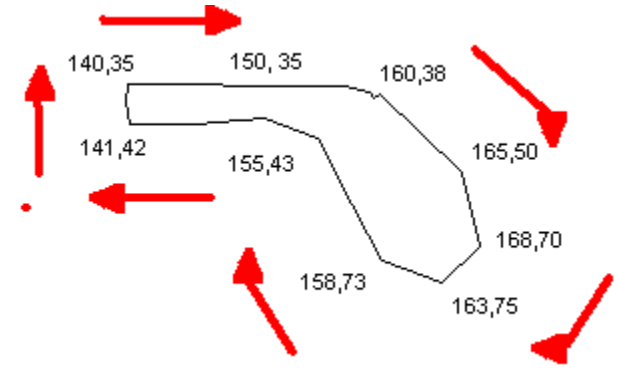
Creating Owner Data Files

The Map data file consists of two main parts; the header and the data. The header record is in the following format:

```
HeaderRecord = record
  ident : string[29];
  detailpos : array [1..32] of longint;
  details : longint;
end;
```

- **`ident`**
  The ident member contains the text **`TWorldMap Component data file.`** The component will check for this value when it loads the map file. If it does not exist an exception will be raised.

- **`detailpos : array [1..32] of longint;`**
  The component will support up to 32 levels of detail of which the **first 4 are reserved.** Level 1 = Country boundaries, 2 = State Boundaries, 3 = Lakes polygon data, 4 = River Data. Values 5 through 31 are left for user defined detail levels. Each array index contains the fileposition of the **start** of that particular detail. This is done for performance reasons, since each level of detail can be handled independantly this prevent scanning the entire file each time a detial is drawn. At a minimum the data file must contain the land mass data. In order to be consistent, the land mass data should be in **POLYGON** format. Polygon format means that the points representing a particular land mass should be organized in sush a manner that they can be drawn consecutively either clockwise or counterclockwise. Windows takes the points and draws them successively. The following example show how to organize points to properly draw a polygon.



  If your points are orgainzed in a top down manner (i.e top point, left point, right point etc..) then it will look like garbage when it is drawn and could potentially blow the GDI. How the polygons are stored will be discussed later. It is important that when creating map data files that all values of this array are initially set to 0. **The system depends on non-used indices being set to 0.**

- **`details : longint;`**
  This member is a flag telling the system how many levels of detial are included in the file.

The data consists of two parts; a polyrecord containing details about the point data and the

point data itself. The polyrecord is a data header record. It contains drawing information about the particular detail, including a description, number of data points, a flag, brush color, brush style, pen color and pen style. The polyrecord declaration is as follows:

```
Polyrecord = record
  description : string[40];
  pcnt : longint;
  flag : longint;
  bcolor : TColor;
  bstyle : TBrushStyle;
  pcolor : TColor;
  pstyle : TPenSTyle;
end;
```

- **description : string[40];**
  The description field is included for ease of maintenance. It is not used in any way within the component, but it allows the developer to associate meaningful descriptions to the data that make file editing much easier. Future releases will allow searches based upon the description.

- **pcnt : longint;**
  The pcnt field is the point count for data that follows this record in the file. This is one of the most important parts of the polyrecord. This should be the number of points, not the zero based value. **In other words DO NOT subtract 1 from the total!** Remember that it is a longint. Attempting to mainpulate this value within Delphi can cause weird things to happen if you try and mix integers with longints.

- **flag : longint;**
  This field is the second most important part of the polyrecord. The first bit (starting from the right) indicates if the point data is to be drawn as a polygon or as line segments. For example, in the map data file, the land masses and lakes are drawn as polygons and the rivers, countries, and states are drawn as line segments. **For polygons the rightmost bit (the ones place) is set to 0, for line segments the bit is set to 1.** To accomplish this in Delphi, examine the following code example:

```
  {if we are using polygon data, set the flag to 0 initially}
  prec.flag := 0;
  {If we want line segments then we set the flag to 1}
  prec.flag := 1;
end;
```

  Additionally, the bit corresponding to the respective level of detail must be 1. For example, if this record was a level 7 detail then bit 7 must be 1. To accomplish this in Delphi, examine the following code example which includes initiallizing the flag:

```
procedure SetFlag(var prec: polyrecord; polydata: boolean; detaillevel :
integer);
var
  flagvalue : longint;
begin
  flagvalue := 1;
  {if we are using polygon data, set the flag to 0 initially}
  if polydata then
    prec.flag := 0
```

```
      {If we want line segments then we set the flag to 1}

      else
        prec.flag := 1;

      {detaillevel is passed as a 7.  To set bit 7 to the on value}

      flagvalue := flagvalue shl detaillevel; {shift left by 7 bits}

      {Now set the flag in the polyrecord}

      prec.flag := prec.flag or flagvalue;{OR the value with the flag}

    end;
```

The flag is important for the system to determine when it is done drawing the detail of the selected level.

- **bcolor : TColor;**
  The bcolor field is of type Tcolor. This value determines which brush color will be used when drawing the detail. For levels 0 thorough 4 the LandBrush, LandPen, BorderPen, LakeBrush, Lake Pen are used and these values are overridden. For levels 5 through 31 these values are set when the particular level draws.

- **bstyle : TBrushStyle;**
  The bstyle field is of type TBrushStyle. This value determines which brush style will be used when drawing the detail. For levels 0 thorough 4 the LandBrush, LandPen, BorderPen, LakeBrush, Lake Pen are used and these values are overridden. For levels 5 through 31 these values are set when the particular level draws.

- **pcolor : Tcolor;**
  The pcolor field is of type Tcolor. This value determines which pen color will be used when drawing the detail. For levels 0 thorough 4 the LandBrush, LandPen, BorderPen, LakeBrush, Lake Pen are used and these values are overridden. For levels 5 through 31 these values are set when the particular level draws.

- **pstyle : TPenSTyle;**
  The pstyle field is of type TPenStyle. This value determines which pen style will be used when drawing the detail. For levels 0 thorough 4 the LandBrush, LandPen, BorderPen, LakeBrush, Lake Pen are used and these values are overridden. For levels 5 through 31 these values are set when the particular level draws.

Following the polyrecord in the file is the point data. The point data, when read into the map component is read into a point array. Important things to remember about the point array. The largest size supported by the worldmap component is 15000 points. This limitation is imposed by the compiler for that omnipresent 64k barrier. I have worked with larger arrays dynamically, but the code gets very messy. Currently no polygons in the mapfile exceed this limitation. (The WIN95 version will not have to worry about this problem).

```
 xypoints : array [0..15000] of Tpoint;
```

The points are stored in the minutes format. (180 degrees corresponds to 10800 minutes). This format was adopted due to ease of use. The X value corresponds to Longitude, the Y value corresponds to Latitude. The allowable ranges for Longitude

are
-10800 (180W) to 10800 (180E). The allowable ranges for Latitude are 5400 (90N) to -5400 (90S).

The following code segment is one way of loading the map file into memory for manipulation

Here are the declarations used:

```
HeaderRecord = record
  ident : string[29]; {TWorldMap Component data file}
  detailpos : array [1..32] of longint;
  details : longint;  {shows number of current detail levels}
end;
Polyrecord = record
  description : string[40];
  pcnt : longint;
  flag : longint;
  bcolor : TColor;
  bstyle : TBrushStyle;
  pcolor : TColor;
  pstyle : TPenSTyle;
end;
TPolyObject = class(TObject)
  description : string[40];
  maxnlat : integer;
  maxslat : integer;
  maxElong : integer;
  maxWlong : integer;
  pcnt : longint;
  flag : longint;
  fpos : longint;
  bcolor : TColor;
  bstyle : TBrushStyle;
  pcolor : TColor;
  pstyle : TPenSTyle;
  ptarray : ^Tpoint;
end;
PTPolyObject = ^TPolyObject;
```

The Tpolyobject is slightly different from the polyrecord in that it maintaints the file position of the data segment and also the data itself in the form of a dynamic array. By loading this into a list it can be manipulated and edited one data segment at a time. The following code loads the File into the list. FWorldDat is a TMemoryStream. Polylist is a Tlist. XYPOINTS is a static array declared for use in indexing individual points. This was done for simplicity due to the complex nature of declarations and calculations of dynamic array indexing (compiling is a pain in the butt if everything is not perfect). This code in NOT complete but illustrates a way to load the file into memory for manipulation.

```
  FWorldDat.Loadfromfile(Opendialog1.filename);
  eofile := FWorlddat.size;
  FWorldDat.Seek(0,0);
  FWorldDat.read(head,sizeof(headerrecord));
  while not Done do begin
    New(mpoly);
```

```
      Mpoly^ := TPolyObject.Create;
      Mpoly^.maxnlat := 0;
      Mpoly^.maxslat := 0;
      Mpoly^.maxelong := 0;
      Mpoly^.maxwlong := 0;
      Mpoly^.fpos := FWorldDat.position;
      pointcnt := 0;
      FWorldDat.read(prec,sizeof(polyrecord));
      if FWorldDat.position = eofile  then begin
        Done := true;
        Dispose(mpoly);
      end else begin
        Getmem(Mpoly^.ptarray,sizeof(Tpoint) * longint(prec.pcnt));
        MPoly^.pcnt := prec.pcnt;
        FWorldDat.Read(Mpoly^.ptarray^,sizeof(Tpoint)* longint( prec.pcnt));
        Mpoly^.flag := prec.flag;
        Mpoly^.description := prec.description;
        Mpoly^.bcolor := prec.bcolor;
        Mpoly^.bstyle := prec.bstyle;
        Mpoly^.pcolor := prec.pcolor;
        Mpoly^.pstyle := prec.pstyle;
        with Mpoly^ do begin
          hmemcpy(@xypoints,@ptarray^,prec.pcnt);
          for i := 0 to  prec.pcnt do begin
            with xypoints[i]  do begin
              if x < 0 then begin
                if x < maxwlong then maxwlong := x;
              end else begin
                if x > maxelong then maxelong := x;
              end;
              if y < 0 then begin
                if y < maxslat then maxslat := y;
              end else begin
                if y  > maxnlat then maxnlat := y;
              end;
            end;
          end;
        end;
        Polylist.Add(Mpoly^);
      end;
    end;
```

To write the file back out to disk uses a similar process. Here is the method I used to recreate the new map data file:

```
procedure TMapEditor.RecreateFile(filename : string);
var
 F : file;
 Hrec : HeaderRecord;
 prec : polyrecord;
 myflag : longint;
 i : integer;
 st,cou,la,ri : boolean;
begin
  myflag := 1;
  st := false;
```

```
  cou := false;
  la := false;
  ri := false;
  Assignfile(F,filename);
  rewrite(F,1);
```

{Initialize the header record and write it to the beginning of the file}

```
  Hrec.ident := 'TWorldMap Component data file';
  Hrec.details := 30; {This is a short cut for the bits 2-4 being on}
  for i := 1 to 32 do Hrec.detailpos[i] := 0;
  Blockwrite(F,Hrec,sizeof(headerrecord));
```

{Iterate through the list and write out each group of data}

```
  for i := 0 to Polylist.count - 1 do begin
    with TPolyobject(polylist.Items[i]) do begin
```

{For each FIRST occurrence of a new level of detail, reocrd the fileposition in the header
record. Notice the hard coded values for the reserved levels of detil in the header record}

```
     if (description = 'STATE BOUNDARIES') and (not st) then begin
         Hrec.detailpos[1] := Filepos(F);
         st := true;
      end else if (description = 'COUNTRY BOUNDARIES') and (not cou) then
begin
         Hrec.detailpos[2] := Filepos(F);
         cou := true;
      end else if (description = 'LAKES') and (not la) then begin
        Hrec.detailpos[3] := Filepos(F);
        la := true;
      end else if (description = 'RIVERS') and (not ri) then begin
        Hrec.detailpos[4] := Filepos(F);
        ri := true;
      end;
```

{Copy the data into the polyrecord from the polyobject }

```
     prec.description := description;
      prec.pcnt := pcnt;
      prec.flag := flag;
      prec.bcolor := bcolor;
      prec.bstyle := bstyle;
      prec.pcolor := pcolor;
      prec.pstyle := pstyle;
      Blockwrite(F,prec,sizeof(polyrecord));
      Blockwrite(F,ptarray^,sizeof(tpoint) * longint(pcnt));
    end;
  end;
  ri := false;
```

{The following code was used to create 2 user levels of detail using rivers and lakes for my data}

```
  for i := 0 to Polylist.count - 1 do begin
    with TPolyobject(polylist.Items[i]) do begin
        if description <> 'RIVERS' then continue;
        if (description = 'RIVERS') and (not ri) then begin
          Hrec.detailpos[5] := Filepos(F);
          ri := true;
        end;
      prec.description := description;
      prec.pcnt := pcnt;
      prec.flag := flag or (myflag shl 5);
      prec.bcolor := bcolor;
      prec.bstyle := bstyle;
      prec.pcolor := clRed;
      prec.pstyle := pstyle;
      Blockwrite(F,prec,sizeof(polyrecord));
      Blockwrite(F,ptarray^,sizeof(tpoint) * longint(pcnt));
    end;
  end;
  ri := false;
  for i := 0 to Polylist.count - 1 do begin
    with TPolyobject(polylist.Items[i]) do begin
        if description <> 'LAKES' then continue;
        if (description = 'LAKES') and (not ri) then begin
          Hrec.detailpos[6] := Filepos(F);
          ri := true;
        end;
      prec.description := description;
      prec.pcnt := pcnt;
      prec.flag := flag or (myflag shl 6);
      prec.bcolor := bcolor;
      prec.bstyle := bstyle;
      prec.pcolor := clRed;
      prec.pstyle := pstyle;
      Blockwrite(F,prec,sizeof(polyrecord));
      Blockwrite(F,ptarray^,sizeof(tpoint) * longint(pcnt));
    end;
  end;
```

{Finally we seek the beginning and rewrite out the header record with the updated data}

```
  Seek(F,0);
  Blockwrite(F,Hrec,sizeof(headerrecord));
  Closefile(F);
end;
```

Important things to remember:

1. Initialize headerrecord detailpos to all zeros before starting.
2. Ensure that the index you use in the headerrecord detail position is the same bit that you turn on for the flag of that same detail level.
3. Remember to set the ones bit to 0 for polygon data and 1 for line segment data.
4. If you are not using alternate reserved levels of detail (i.e. you have no river, lake,state or country data in your file) then set the empty areas with the fileposition of the last data in the file. For example if you have polygon data and then have all user defined elements such as roads or county boundaries your header record would be:

detailpos[1] := 0 {file position of the end of the polygon data the start of the country data}
detailpos[2] := 0 {repeat the process.. this way the file will skip this area since the component
detailpos[3] := 0 {draws from 1 to 2, 2 to 3 etc}
detailpos[4] := 0
detailpos[5] := 143255 {your data starts here with the 5th bit set in the flag plus a 0 or 1 depending on
detailpos[6] := 283255 {data type of polygon or line segment, 6 can be here or can be a 0}
detailpos[6] := 0;

This is a small example of the owner data files. A more robust example will be included with the final release.

Drawing Directly On the Map Canvas

Another extensible capability of the TWorldMap component is the ability to directly draw on the map canvas. This allows the programmer to use the map as a backdrop for a wide range of custom drawing operations. The following example illustrates how to allow interactive line drawing on the map canvas to graphically connect two points on the map and generate a line that is stored in the component. The drawing operation is accomplished by depressing the mouse and the start point, dragging to the end point and releasing the mouse button. The operation is broken into 4 steps: capturing the point of start, capturing the drawing operation, capturing the endpoint, and adding the new line to the component. To accomplish this we need some class variables to keep our status as we move through the process.

```
    FSPt : Tpoint; {start point}
    FEPt : Tpoint; {end point}
    FbDrawingEnabled : boolean; {flag says we are ready
                                 to draw}
    FbDrawingLine : boolean; {flag says we are drawing}
```

At the core of the interactive drawing is the DrawLine procedure. The drawline procedure does all of the work. If the IsFinal parameter is false we use NOTXOR to erase the last line and redraw the new line. When IsFinal is true, we copy the line to the canvas.

```
procedure TMapForm.DrawLine(X,Y:integer;IsFinal : boolean);
var
  MyCanvas : TCanvas;
begin
   MyCanvas := Worldmap1.GetCanvas;
   with MyCanvas do begin
      if not isfinal then
  { use to Raster Op codes to make the line eraseable}
        SetROP2(Handle,R2_NOTXORPEN)
      else
        SetROP2(Handle,R2_COPYPEN);
      Pen.Style := pssolid;
      Pen.Color := clRed;
      Pen.width := 1;
      Moveto(FSPt.X,FSPt.Y);
      Lineto(FEPt.X,FEPt.Y);
      Moveto(FSPt.X,FSPt.Y);
      Lineto(X,Y);
      FEPt.x := X;
      FEPt.Y := y;
  end;
end;
```

Once we have decided to start the drawing process, we need to set the fbDrawingEnabled class global variable to true. This tells us that when we depress the mouse button we are starting our line drawing operation. We change the cursor to indicate we are ready to draw, set the FbDrawingEnabled to true, and turn off our pointclick events to prevent any interference. Here is the menu event:

```
procedure TMapForm.DrawLine1Click(Sender: TObject);
begin
  Worldmap1.cursor := crPencil;
```

```
  Worldmap1.EnablePointClick := false;
  FbDrawingEnabled := true;
end;
```

The next step in directly drawing on the map is to capture the starting point. We use the OnMapMouseDown event to start this process. We check the FbDrawingEnabled, and if true, we prepare to draw the line. We turn off drawing enabled to prevent resetting our starting points, set the FbDrawingLine variable to true to indicate that we are now in the process of drawing, and initialize our start and end points with the current mouse position.

```
procedure TMapForm.WorldMap1MapMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer; Lat,
  Long: Longint);
begin
   if FbDrawingEnabled then begin
      Worldmap1.showlines:=true;
      FbDrawingEnabled := false;
      FbDrawingLine := true;
      FSPt.X := x;
      FSPt.y :=y;
      FEPt.x := x;
      FEPt.y := y;
   end;
end;
```

In order to track the line drawing operation we need to capture any mouse movements while the FbDrawingLine is true. To accomplish this, the MapMouseMove event is used. Note the FALSE parameter sent to DrawLine. This keeps the line erasable until we are ready to finalize it.

```
procedure TMapForm.WorldMap1MapMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer; Lat, Long: Longint);
begin
 if FbDrawingLine then begin
   DrawLine(x,y,FALSE);
 end;
end;
```

Finally, when we are ready to record our line, we need to capture the MapMouseUp event to finish drawing the line. We call Drawline with IsFinal set to true to finalize it. We call the convert methods to get the lat/long of our stored X,Y points and add the line to the component. Note the **div** 60 in the AddLine method, because the convert method returns seconds.

```
procedure TMapForm.WorldMap1MapMouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer; Lat, Long: Longint);
var
 ilatf,ilongf,ilatt,ilongt : longint;
begin
  if FbDrawingLine then begin
     DrawLine(x,y,TRUE);
     FbDrawingLine := false;
     Worldmap1.cursor := crCross;
     With worldmap1 do begin
       ConvertXYtoLatLong(ilatf,ilongf,FSPt.X,FSPt.y);
       ConvertXYtoLatLong(ilatt,ilongt,FEPt.X,FEPt.y);
       AddLineInt(ilatf div 60,ilongf div 60,ilatt div
```

```
                60,ilongt div 60,clRed,2,0);
        EnablePointClick := true;
        Showlines := true;
    end;
    ShowLines1.checked := true;
  end;
end;
```

This is only one way of directly drawing a line on the canvas. Other methods might include using the shiftstate, or a different mouse button instead of a global class variable. Additionally, the developer can create their own featurelist and manage drawing it with the OnPaint method of the Map component depending on changing conditions.

Hooking Database Events to PointClicked Events

One of the powerful features of the TWorldMap component is the ability to click on points shown on the map and take actions based upon the event. One of the more common uses would be to show minimal information on the map with the ability to display detail upon request. To accomplish this, a unique or identifying piece of information must be stored with the point to allow a parameterized query to be executed based upon the unique information. The following example uses the tag value as the unique value. The data used for this example has a unique record id. When the points are added to the component, the tag value is set to the record id. When an OnPointClicked event is triggered, the tag value is used to retrieve additional information from the database. Since the entire point is passed to the event, any data element stored within the point can be used to take an action. Using different shapes and colors would allow be a way to determine what action to take on the click event.

```
procedure TMapForm.WorldMap1PointClicked(Sender: TObject; Curpoint:
TPointObject; AIndex: Integer);
var
  workstr : string;
  I       : integer;
  cwd     : string;
begin
   screen.cursor := crHourglass;
   GetDir(0,cwd);
   workstr:= 'Current Point ID is '+
             inttostr(Curpoint.tag) +#13;
   workstr:= workstr+Curpoint.labels.strings[0]+#13;
   try
     With Clickqry do begin
       databasename := cwd;
       params[0].AsInteger := Curpoint.tag;
       prepare;
       open;
       Workstr := workstr +
            Fieldbyname('prov_name').Asstring+#13;
       WorkStr := workstr +
             Fieldbyname('country_name').Asstring;
     end;
     screen.cursor := crDefault;
     messagedlg(workstr,mtInformation,[mbOk],0);
   finally
     Clickqry.close;
     Screen.cursor := crDefault
   end;
end;
```

Attaching User Data to Points and Lines

Another powerful feature associated with the lines and points is the data pointer. The data pointer allow user data to be attached to each point. Since the pointer is generic, any object can be attached, as well as normal pascal data structures. The following example is a simple illustration of attaching a record to a point.
A simple record and a pointer to that record is declared in the type section.

```
mydata = record
  anInt : integer;
  AString : string
end;
Pmydata = ^mydata;
```

In the method used to update the point, a new pointer is created, and the record contents are filled and assigned. The record is then attached to the data field.

```
var
  MyPoint : TPointObject;
  pmd : pmydata;
begin
  MyPoint := TPointObject.Create;
  try
    Worldmap1.GetPoint(MyLine,4);
    New(pmd);
    pmd^.anInt := 3;
    pmd^.AString := Test String;
    Mypoint.data := Pointer(pmd);
    MyPoint.pointstyle := psSquare;
    MyPoint.color := clgreen;
    WorldMap1.UpdatePoint(Mypoint,4,FALSE);
  finally
    MyPoint.Free;
  end;
end;
```

Glossary

coordinates, origin of
Points in a system of coordinates which serves as a zero point in computing the system's elements or in prescribing its use.

Degree
Unit of measurement for map system. 1 Degree = 60 miles.

grid
Network of uniformly spaced parallel lines intersecting at right angles. When superimposed on a map, it usually carries the name of the projection used for the map- that is, Lambert grid, transverse Mercator grid, universal transverse Mercator grid. The TWorldMap component uses a Mercator Grid.

latitude
Angular distance, in degrees, minutes, and seconds of a point north or south of the Equator.

longitude
Angular distance, in degrees, minutes, and seconds, of a point east or west of the Greenwich meridian or Prime meridian.

minute
A unit of measurement used in map systems, 1 minute = 1 mile.

prime meridian
Meridian of longitude 0 degrees, used as the origin for measurements of longitude. The meridian of Greenwich, England, is the internationally accepted prime meridian on most charts. However, local or national prime meridians are occasionally used.

scale
The scale determines the physical dimensions of the map viewing area. In nautical terms a minute represents 1 mile, since a Degree is 60 minutes it is also 60 miles. Specifying a scale of 600 miles would yield a display of 10 degrees wide by 5 degrees high. (Remember the 2:1 aspect ratio).

second
A second is a unit of measurement used in map systems. ! second = 1/60th of a mile.

Demo Help

The demo menu and tool bar attempts to demonstrate some of the built in capabilities of the component. To get more information about an area move the cursor over it. If the cursor
turns into a hand, it is a link to information about that area.

TWorldMap Reference
Component Installation
Component Limitations
Technical Support
Glossary of Terms Used

Map Scale

The map scale shows the width of the current map view in minutes (nautical miles). This field is updated whenever the map is zoomed in or out. This value is calculated using the difference between the StartELongitude and StartWLongitude properties

Show Rivers

The show rivers button toggles (turns off and on) the rivers detail option of the map. Click this button to turn on the rivers, or if the rivers are showing, click this button to turn the rivers off.

Zoom In By Factor

The zoom in by factor button uses the components ZoomInByFactor Method to zoom the map by a factor of 2. After clicking the button, the cursor will change to the magnifying glass. Click the point on the map to become the center of the newly zoomed area. This takes effect regardless of the current map view. If you have used the Zoom In button and previously zoomed the map, the factor will increase the current zoom factor by 2.

Zoom Out By Factor

The zoom out by factor button uses the components ZoomOutByFactor Method to zoom the map out by a factor of 2. The button automatically zooms the map out maintaining the current center.

Zoom Factor

This field shows the current zoom factor of the map.

Show Grid

The show Grid button toggles (turns off and on) the Grid detail option of the map. Click this button to turn on the Grid, or if the Grid is showing, click this button to turn the Grid off. When the Grid is showing the menu item Adjust Grid allows you to change the spacing of the Grid lines

Show States/Provinces

The show states button toggles (turns off and on) the states detail option of the map. Click this button to turn on the states/provinces, or if the states/provinces are showing, click this button to turn them off.

Show Lakes

The show lakes button toggles (turns off and on) the lakes detail option of the map. Click this button to turn on the lakes, or if the lakes are showing, click this button to turn the lakes off.

Show Points

The show points option demonstrates the components capability to store and show points. If there are no points loaded, the demo loads the national capitals table using the AddPointDec method. The Demo uses a modulo number to generate the point colors, and displays the points. If the points are showing the points are turned off by setting the ShowPoints property to false. The colors are used to demonstrate the user defined OnPointFilter event.

Show Lines

The show lines option demonstrates the components capability to store and show lines. The ShowLines property controls the showing of lines if there are lines stored within the component. To add a line to the component, you can use the menu option DrawLine to interactively draw on the canvas or use the Plotline menu option to enter the coordinates of the line.

Related Topics

AddLine
DeleteLine
ClearLines
GetLine
UpdateLine

Show Labels

The Show Labels option toggles (turns on and off) the labels associated with the points loaded in the component. If points are loaded, labels can be shown with or without the points. This option uses the ShowPointLabels property

Related Topics
ShowPoints
AddPoints
ClearPoints

Latitude

This edit box demonstrates one of the built in capabilities of the component. Whenever the EnableUpdatedPoints property is true, the CurrentLatitudeStr, CurrentLatitude, CurrentLongitudeStr and CurrentLongitude properties are continuously updated. The OnPointChange event is triggered each time the cursor is moved within the map area. Keeping the edit box current with the mouse position takes one line of code.

Longitude

This edit box demonstrates one of the built in capabilities of the component. Whenever the EnableUpdatedPoints property is true, the CurrentLatitudeStr, CurrentLatitude, CurrentLongitudeStr and CurrentLongitude properties are continuously updated. The OnPointChange event is triggered each time the cursor is moved within the map area. Keeping the edit box current with the mouse position takes one line of code.

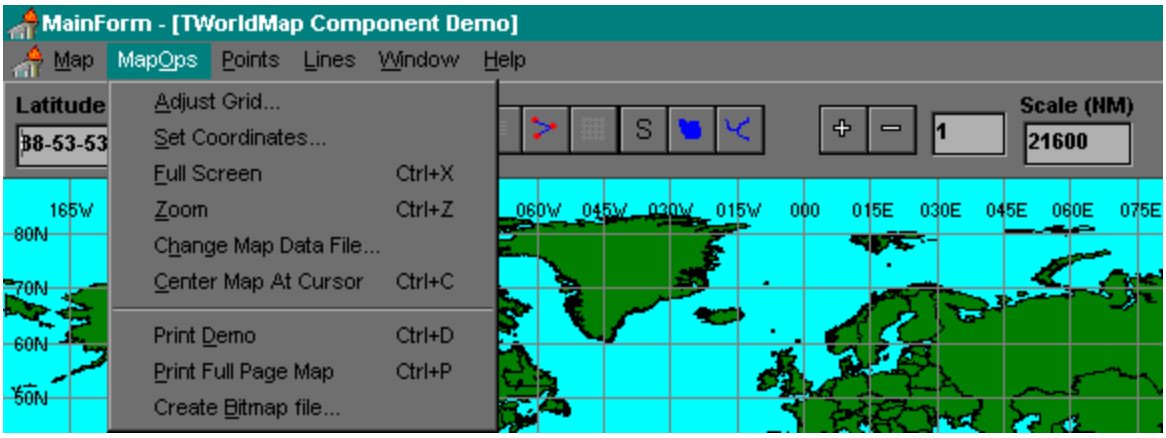
Zoom In

The Zoom In option demonstrates another built in capability of the component. After clicking the Zoom In option, the EnableZoom property is set to true. While the component is in this state, the cursor changes into the zoom cursor (Magnifying Glass) when the cursor is moved over the map. Holding the left mouse button down and dragging creates a zoom rectangle. When the mouse is released, the map zooms to the area enclosed by the rectangle. Upon completion of this operation the enablezoom property is reset to false.

Full Screen

This option returns the map to full screen
using the FullScreen method.
Setting the map to full screen
updates the the StartELongitude, StartWLongitude,
StartNLatitude,StartSLatitude properties.

MapOps Options



TWorldMap Reference
Component Installation
Component Limitations
Technical Support
Glossary of Terms Used

Adjust Grid...

The Adjust Grid is an interface to the GridGapLat and GridGapLong properties which determine the space between the vertical and horizontal lines on the map. Changing these properties at runtime allow the grid to provide higher resolution as the map is zoomed.

Related Topics
DetailOptions

Set Coordinates...

The SetCoordinates Dialog allows the user to use the SetCoordinates Method of the component. This method provides another way to zoom the map and center it at a point. When executed the map adjusts to the new scale centered around the lat / long passed.

Related Topics
SetCoordinatesDec
SetCoordinatesInt

Change Map Data File

The Change Map Data File option allows a new map file to be loaded into the component at runtime by changing the Mapfile property. The map automatically reloads the new file when it attempts to draw. This allows user created data files with varying levels of detail to be created and shown on the fly by the map component. The release version of the map component comes with three data files, WorldMapL1.dat, WorldmapL2.dat and WorldMapL3.dat. Level 1 is the highest detail and level 3 is the lowest.

Center Map At Cursor

The Center Map at Cursor option demonstrates a special use for the ZoomInByFactor Method. By pasing a 1 as the factor, the method will recenter the map on the passed coordinates. This allows the map to be panned in any direction.

Related Topics
ZoomOutByFactor

Print Demo

This options demonstrates the components capability to print the map at a chosen location multiple times on the same page with different attributes. This option uses the PrintMap method

Related Topics
SaveToFile

Print Full Page Map

This option uses one line of code to print the map on a full page. This option also uses the PrintMap method

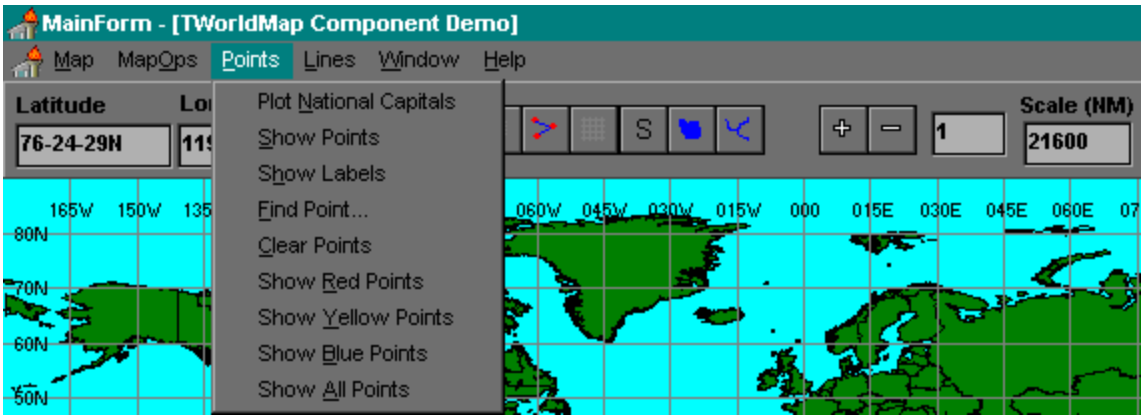
Related Topics
SaveToFile

Create Bitmap File

This option displays a dialog box allowing the user to specify the dimensions of a bitmap file that the component creates. This BMP file can be used in any program that use bitmaps. This option uses the SaveToFile method

Related Topics
PrintMap

Points Options



TWorldMap Reference
Component Installation
Component Limitations
Technical Support
Glossary of Terms Used

Plot National Capitals

If there are no points loaded, the demo loads the national capitals table using the AddPointDec method. The Demo uses a modulo number to generate the point colors, and displays the points.

Related Topics
AddPoint
AddPointInt
DeletePoint
ClearPoints
GetPoint
UpdatePoint

Find Point

The find point demonstrates the components built in capability to search the internal pointlist for a particular value in a point object. The fields of the pointobject correspond to a Findtype, and the FindPoint Method executes the search

Related Topics
GetPoint
UpdatePoint

Clear Points

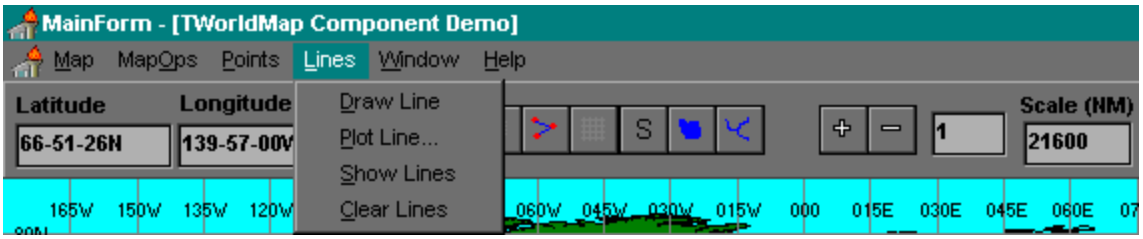
The Clear Points option deletes all points stored within the component using the Clearpoints method

Related Topics
DeletePoint

Show Colored Points

The Show Red Points, Show Yellow Points, Show Blue Points and Show All Points options demonstrate the OnPointFilter capability of the component. After plotting the National Capitals, the points can be selectively shown by clicking on the menu color option.

Lines Options



TWorldMap Reference
Component Installation
Component Limitations
Technical Support
Glossary of Terms Used

Draw Line

Draw line demonstrates the ability to draw directly on the map canvas and record information relative to the latitude and longitude of the map. Selecting this option turns the cursor into a pencil. Move the mouse cursor to where you would like a line to begin. Hold the mousebutton down and drag the cursor to the location you would like the line to end and release the button. A line has now been added to the components internal line list. The line will show each time the map is redrawn if the showlines property is true.

Related Topics

OnMapMouseMove
OnMapMouseDown
GetCanvas
OnPaint
OnMapMouseUp
AddLine
Drawing Directly on the Map Canvas

Plot Line

The plot line is a more conventional way to add a line to the map. This dialog allow you to enter a starting point and an ending point for a line. The line is added to the components line list and is shown anytime the map is drawn.

Related Topics
AddLine
AddLineInt
AddLineDec

Clear Lines

The Clear Lines option deletes any previously added lines, using the components ClearLines Method

Related Topics
DeleteLine