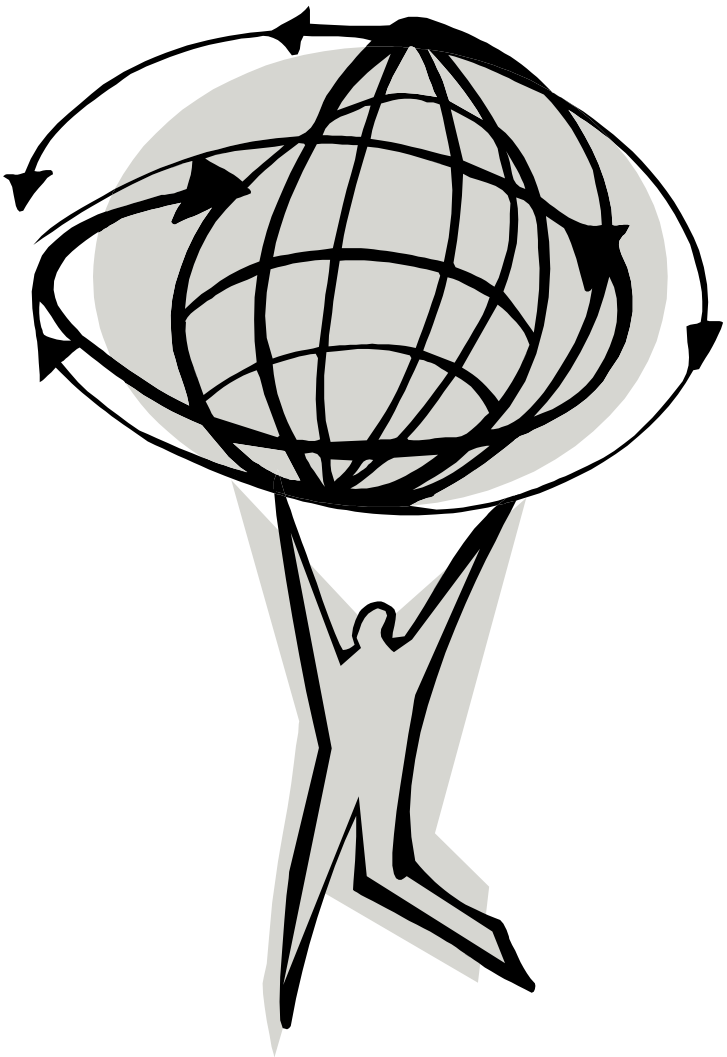
UnderTow Software

MapOCX Pro v7.1

Release 2

Reference Manual



Undertow Software, Corp. - PO BOX 249, N. Andover, MA 01845
Ph (978) 794-9377 Fx (978) 688-6312 www.undertowsoftware.com

Undertow Software Corp.
PO Box 249
N. Andover, MA 01845
U.S.A.
World Wide Web http://www.undertowsoftware.com

MapPro OCX, ActiveX Mapping Control for Windows, Version 7.1, Rel. 2, User's Manual

Copyright © 2006 by Undertow Software Corp.

All rights reserved. Printed in the United States of America. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

TABLE OF CONTENTS

INTRODUCTION 15
Map Data 15
Stock Dialogs, Toolbars,… 15
Map Rotation 15
Underlays & Overlays 16
Geocoding and Reverse Geocoding 16
Route Optimization 16
User Registration 17
What's Included in the Developers Toolkit 17
Minimum Requirements 17
Coordinate System 17
Installing and Using the Control 18
Properties, Methods, Events 19
Internal Data Structure 19
Default Mouse Actions 19
Left Click (Re-centering) 19
Right Click (Zooming Out) 19
Left Click & Drag (Zooming In) 19
Shift & Left Click (Quick Routing) 19
Control & Left Click (Attributes) 20
Files in this package… 21
Root Directory of Distribution CD-ROM/DVD 21
Data1 Directory of Distribution CD-ROM/DVD 21
Data2 Directory of Distribution CD-ROM/DVD 21
Data3 Directory of Distribution CD-ROM/DVD 21
States Directory of Distribution CD-ROM/DVD 21
Data4 Directory of Distribution CD-ROM/DVD (if POI data is Licensed) 22
Underlays Directory of Distribution CD-ROM/DVD 22
Sample Source Code Directory of Distribution CD-ROM/DVD 22
Development Considerations – Quick How To's and things to be aware of 23
Rotated Maps 23
Shading 23
Sample Code 23
Searching for Places & Geocoding 23
Reverse Geocoding 23
Displaying Data Points 23
Adding your own streets 23
Single Instantiation – Many Map Views 24
Distribution Files 24
*.SIG Files 24
Upgrading from an earlier Version 24
Street Level Data Visibility Problems 24
Drawing on the Maps 25
Routing 25
World Map Extents 25
MapOCX Pro Licensing 25
Technical Support 26

Updates 26
OCX Reference Section 27
AboutBox() Procedure 27
AddViaPoint(x,y:Double; s:String) Procedure 27
AutoConfig:Boolean Property 27
AutoPaint:Boolean Property 28
AutoQuery:Boolean Property 28
BevelInner:Integer Property 29
BevelOuter:Integer Property 29
BevelWidth:Integer Property 30
BorderWidth:Integer Property 30
CAD:Interface Interface 30
CenterView(X,Y:Integer) Procedure 30
ClearExclusion() Procedure 31
ClearStreets() Procedure 31
ClearOverlay() Procedure 31
ClearViaPoints() Procedure 32
ClrStrBubble() Procedure 32
Color:Integer (or enumerated ColorName) Property 32
Coords:Integer Property 32
Cursor:Integer Property 33
Custom:Integer Property 34
DataSource:txDataSource Property 34
DbFile:String Property 35
DegFormat:TxDegFormat Property 35
DeleteAllitems() Procedure 36
DeleteItem(ID:Integer) Procedure 36
DeleteItemRange(ID1,ID2:Integer) Procedure 37
DeleteViaPoint(ID:Integer) Procedure 37
DevCode:String Property 37
DevPass:String Property 38
DirectDraw(dc, offX, offY, Wd, Ht : Integer, Clear, Scale, BW : Boolean) Procedure 38
Distance(x1,y1,x2,y2: Variant):integer Function 39
DirectView(dc, w, h:integer; var LonL, LatB LonR, LatT: Double; var DvScale:Double) 40
DMS(x: OleVariant): string Function 41
DrawBubble(dc:integer; x,y:integer; S:string) Procedure 41
DrawLine(dc:integer; x1,y1,x2,y2:OleVariant;w,cl,mode: Longint) Procedure 42
DrawNorth(dc,x,y:integer) Procedure 43
DrawObject(dc:integer,Lon,Lat:OleVariant;Item,Color: Longint) Procedure 44
DrawScalebar(dc, x, y: Integer) Procedure 44
EdgePan:Boolean Property 45
EdgePanAmount:Double Property 45
EdgePanWidth:Integer Property 45
Enabled:Boolean Property 46
ExecClosest() Procedure 46
ExecDbImport() Procedure 47
ExecLonLat() Procedure 47
ExecMem(X:LongInteger):LongInteger Function 48
ExecMethod(s:String) Procedure 48
ExecPhone() Procedure 51
ExecPlace() Procedure 51

ExecPrint() Procedure 52
ExecPrintEX() Procedure 54
ExecRegister(x:LongInt) Procedure 54
ExecRoadOption() Procedure 55
ExecRoute(x1, x2, x3, x4: Double) Procedure 57
ExecSearch(Title:string;Option:Integer) Procedure 61
ExecStreet() Procedure 65
ExecZipcode() Procedure 66
FindAcEx(temp:String) Procedure 66
FindCity(temp:String) Procedure 67
FindClosest(X,Y,Radius:OleVariant) Procedure 69
FindClosestPlace(X,Y,Rad,Pop,Opt):string Procedure 71
FindItem(X,Y:OleVariant) Function 72
FindFirstItem(X,Y:OleVariant):String Function 73
FindNextItem():String Function 73
FindItemclose() Method 74
FindRoute(Lon1,Lat1,Lon2,Lat2:OleVariant;Opt: LongInt) Procedure 74
FindViaRoute(Opt: LongInt) Procedure 77
FindStr(const t, t2: string;add: Integer;xctr, yctr, mradius: OleVariant;const ststr: string) 78
FindZip(z:LongInt) Procedure 80
FirePmapEvent(EventID, Delay:Integer) Method 81
Font:String Property 82
GeoFind(s:String):Integer; Function 82
GeoFindArray(S:strring); Method 86
GeoFindClose() Procedure 88
GeoFindFirst(GeofindString:String):String Procedure 88
GeoFindNext():String Procedure 89
GeoFindParse(Field, Result:String):String Procedure 89
GetProductCode:String Property 90
GetRouteFirst(x1,y1,x2,y2:double; Consolidate:boolean; Option:integer):String 90
GetRouteNext():String Procedure 91
GetRouteClose() Procedure 92
RouteParse(Fld:Variant, s:string):String Procedure 92
GetViaRouteFirst(Consolidate:Boolean; Options:Integer):String Procedure 93
GetViaRouteNext():String Procedure 94
GetViaRouteClose():String Procedure 94
GotoPoint(x, y: OleVariant) Procedure 94
Grid:Boolean 94
Handle:Integer Property 95
HeadsUp(x1,y1,x2,y2:OleVariant):longint Function 95
HelpPath:String Property 96
HideAllItems() Procedure 96
Import:ImportLayer; Property 97
ImportMgr:ImportManager; Property 97
InitNonVis() Procedure 97
Int2Lat(i:Integer):Double Function 97
Int2Lon(i:Integer):Double Function 98
IsDrawing:Boolean Property 98
ItemFontSize:Integer Property 99
Kms Property:Double 99
Lat2Int(x:OLEVariant):Integer Function 99

LatBottom:Double Property 100
LatCenter:Double Property 100
LatTop:Double Property 100
LL2INT(x,y:OLEVariant):LongInt Function 101
LLMode:Integer Property 101
Load_Air:Boolean Property 102
Load_City:Boolean Property 102
Load_County:Boolean Property 102
Load_Highway:Boolean Property 103
Load_Hydro:Boolean Property 103
Load_Landmark:Boolean Property 103
Load_Park:Boolean Property 104
Load_Shore:Boolean Property 104
Load_State:Boolean Property 104
Load_World:Boolean Property 104
LoadConfig(path:String) Procedure 105
Loaded:Boolean Method 105
LoadExclusion(filename:String) Procedure 106
LoadStreets(s:String) Procedure 106
LoadViaFile(s:String) Procedure 106
Lon2Int(x:OLEVariant):Integer Function 106
LonCenter:Double Property 107
LonLatStr(x,y:OLEVariant):String Function 107
LonLeft:Double Property 107
LonRight Property 107
Magnitude:Integer Property 108
MainLay:Boolean Property 108
MapCount:LongInt Property 109
Mapmode:Integer Property 109
MapUnits:Integer Property 112
Miles:Double Property 112
OnClick Event 112
OnDblClick Event 113
OnDirect Event 113
OnDirectBefore Event 114
OneWayColor:Integer Property 114
OneWayShow Property 115
OneWayUse Property 115
onCADChange(Current:Long) Event 115
OnFind Event 116
OnFindDir Event 117
OnFindPlace Event 117
OnFindRte Event 118
OnMouseDown Event 118
OnMouseMove Event 118
OnMouseUp Event 119
OnOptiRouterMsg Event 120
OnPaintAfter Event 120
OnPmapEvent Event 121
OnPaintBefore Event 121
OnResize Event 122

OnUnderAfter Event 122
OnUnderBefore Event 122
OnStatus Event 122
OpenOverlay(s:String) Procedure 123
OptiRouter:Interface Interface 123
OptiRouterBtn:Boolean Property 123
OptiPathOnTop:Boolean Property 124
Overlay:Boolean Property 124
Path_Data1 Property 124
Path_Data2 Property 124
Path_Data3 Property 125
Path_Data4 Property 125
Path_Library Property 125
Path_States0:String Property 125
Path_States1:String Property 126
PhoneRegInfo Property 126
PmPalette:Integer Property 126
PmParent:Integer Property 127
PmScale:Double Property 127
POIMgr:Interface Property 127
PopUpRoute:Boolean Property 128
PostUnderlay:Integer Property 128
QueryObj(x,y:Double) Property 129
Redirty Procedure 129
Redraw Procedure 130
Refresh Procedure 131
ResizeCtl(dx,dy:Integer) Procedure 131
Result Property 131
RoadOption(Option, Atrib:LongInt) Procedure 132
Rotate(x:Double) Procedure 134
RouteProgressIcon(FineName:string):Boolean Procedure 134
RoutingActive:boolean Property 134
SaveConfig Procedure 135
SaveExclusion(s:String) Procedure 135
SaveStreets(s:String) Procedure 136
SavetoBitmap(s:String) Procedure 136
SavetoGif(s:String) Procedure 136
SaveViaFile(s:String) Property 137
SaveView:Boolean Property 137
Scale Property:Double 137
ScaleBar:Integer Property 138
Screen_Aspect:Double Property 138
SetDirtyRect(LeftX,TopY,RightX,BottomY) Procedure 138
SetExclusion(IDString:string; x1,y1,x2,y2:double) Procedure 139
SetItem(id: Integer; x, y: OleVariant) Procedure 140
SetItem2Back(id:Integer) Procedure 141
SetItem2Front(id:Integer) Procedure 141
SetItemAngle(id:integer;x:Double) Procedure 141
SetItemBitmap(id:integer; handle:integer) Procedure 142
SetItemLocalBitmap(id:integer; s:String) Procedure 143
SetItemString(id:integer; s:String) Procedure 143

SetItemVis Procedure....144
SetOption(OpCode, Option:LongInt):LongInt Function....144
OpCode = 00 (Special Color Options)....145
OpCode = 01 (Background Color)....146
OpCode = 02 (Water Color)....146
OpCode = 03 (Parks Color)....146
OpCode = 10...1F (Build-in Color Shades)....146
OpCode = 20...5F (State Colors)....146
OpCode = $80 (Major Street Labeling)....147
OpCode = $81 (Minor Street Labeling)....147
OpCode = $82 (Global Street Labeling)....147
OpCode = $83 (Street Label Interference)....147
OpCode = $84 (Hide Suffix)....147
OpCode = $85 (Street Text Size)....147
OpCode = $86 (User Item Visibility)....147
OpCode = $88 (Display Highway Shields)....147
OpCode = $89 (Display Coasre Layer)....147
OpCode = $8F (Display Bounding Polygons)....148
OpCode = $90 (State Text Size)....148
OpCode = $91 (Place Name Text Size)....148
OpCode = $92 (Landmark Text size)....148
OpCode = $93 (Reserved)....148
OpCode = $94 (Highway Shield Text Size)....148
OpCode = $95 (Street Name Text Size)....148
OpCode = $98 (Street Data Threshold)....148
OpCode = $99 (City Label Spacing)....148
OpCode = $100+N (Layer Visibility)....148
OpCode = $200 (State Polygon Shading)....150
OpCode = $201 (County Polygon Shading)....150
OpCode = $202 (MCD Polygon Shading)....150
OpCode = $203 (Place Polygon Shading)....150
OpCode = $301 (State Boundary Color)....150
OpCode = $302 (County Boundary Color)....150
OpCode = $303 (MCD Boundary Color)....150
OpCode = $C001 (Debug Mode)....150
OpCode = $C002 (Render Time)....150
OpCode = $C003 (Load Data Time)....150
OpCode = $C004 (World Extents)....150
OpCode = $D001 (Memory Usage)....150
OpCode = $D002 (Memory Page Size)....150
OpCode = $D003 (Free Memory)....151
OpCode = $D004 (Max Memory)....151
OpCode = $D005 (One Way Arrow Collision)....151
OpCode = $D007 (Min Zoom Scale)....151
OpCode = $D008 (Grid File Memory Usage)....151
OpCode = $EEE2 (Internal Bitmap Mode)....151
OpCode = $EEE4 (User Item Paint Order)....151
OpCode = $EEE9 (Double Lined Roads)....151
OpCode = $EEEA (Routing Local Radius)....151
OpCode = $EEEB (Routing Band Width)....152
OpCode = $EEEC (Routing Highlight Resolution)....152

OpCode = $EEEE (FindClosest Options) 152
Shade_Cnty:Boolean Property 154
Shade_MCD:Boolean Property 154
Shade_Plc:Boolean Property 155
Shade_State:Boolean Property 155
ShowAllItems() Procedure 155
ShowToolBar() Procedure 156
StartView(s:String) Property 157
Street:string Property 157
Synch:Boolean Property 158
Texture:Boolean Property 158
TitlePrint(s:String) Property 158
TitleUser(s:String) Property 159
ToolBarMode:Integer Property 159
Underlay:Boolean Property 159
UnderlayFile(s:String) Property 160
UnderlayTransparent:Boolean Property 161
UnderlayTrColor:Integer Property 161
ViaCount:Integer Property 161
ViewCmd(s:String) Property 162
Visible:Boolean Property 162
Xcord:Double Property 162
Ycord:Double Property 163
ZoomAll() Procedure 163
ZoomCan() Procedure 163
ZoomIn() Procedure 164
ZoomIWindow(x1, y1, x2, y2: Interger) Procedure 164
ZoomLast() Procedure 164
ZoomOut() Procedure 165
ZoomOverlay() Procedure 165
ZoomPan(i:Integer) Procedure 165
ZoomSP(s:String) Procedure 166
ZoomUnderlay() Procedure 167
ZoomWindow(x1, y1, x2, y2: OleVariant) Procedure 167
.CAD Interface 168
.CAD.Arrow(x1,y1,x2,y2:Double):CadObj Method 169
.CAD.Bezier(X1,Y1,X2,Y2,X3,Y3:Double):CadObj Method 169
.CAD.BringToFront Method 170
.CAD.Brush Property 171
.CAD.Clear() Method 171
.CAD.Count:Integer Property 172
.CAD.Delete():Integer Method 172
.CAD.Ellipse(X1, Y1, X2, Y2:Double):CadObj Method 172
.CAD.Extents :TExtentRec Method 173
.CAD.Font() Method 173
.CAD.GetMarker(n:Integer):LongInteger Method 174
.CAD.GetMetaObj(s:String):LongInteger Method 174
.CAD.GetSymbol(index:Integer):LongInteger Method 174
.CAD.Group() Method 175
.CAD.ImportFile() Method 175
.CAD.LoadFromFile(s:String; Option:Integer) Method 175

.CAD.Marker(X, Y:Double; hnd:Integer) Method 176
.CAD.mCircle(Xc, Yc, Xp, Yp, Aspect:Double):CadObj Method 176
.CAD.MetaObj(X1,Y1,X2,Y2:Double; Hnd:LongInteger):CadObj Method 176
.CAD.mLine(X1, Y1, X2, Y2:Double):CadObj Method 177
.CAD.Objects(n:integer) array of CadObj Method 177
.CAD.ObjectType:Integer Property 178
.CAD.Pen() Method 179
.CAD.Polygon(Points:TRPoint; N:Long):CadObj Method 180
.CAD.Polyline(Points:TRPoint; N:Long):CadObj Method 180
.CAD.Rectangle(X1, Y1, X2, Y2:Double):CadObj Method 181
.CAD.RegularPolygon(Xc, Yc, Xp, Yp:Double; N:Integer):CadObj Method 181
.CAD.Rotate(Xp, Yp, Angle:Double) Method 182
.CAD.SaveToFile(S:String) Method 182
.CAD.ScaleToSize(Value,Height:Double):Double Function 183
.CAD.SelectRange(Ns,Nn,Option:Integer) Method 183
.CAD.SelectRect(x1,y1,x2,y2:Double, Option:Integer) Method 184
.CAD.SendToBack Property 185
.CAD.Symbol(X1,Y1,X2,Y2:Double; Hnd:LongInteger):CadObj Method 186
.CAD.Text(X,Y: Double; s:String) Method 186
.CAD.TextBubble(X,Y:Double, S:String) Method 186
.CAD.TextFloat(X1,Y1,Size:Float; Caption:String):CadObj Method 187
.CAD.Toolbar Interface 187
.CAD.Ungroup() Method 188
.CAD.Visible Method 188
CAD Object Properties 188
.objects().Brush Property 188
.objects().Caption Property 189
.objects().Font Property 190
.objects().GreatCircle:Boolean Property 191
.objects().MoveAbs(X,Y: Double) Property 192
.objects().MoveRel (X,Y: Double) Property 192
.objects().ObjectType:Integer Property 192
.objects().Pen Property 193
objects().Selected:Boolean Property 194
.objects().Tag:Integer Property 194
.objects().Visible:Boolean Property 194
Visual CAD Toolbar Interface 195
CAD Attribute Dialog 200
.OptiRouter Interface 204
.OptiRouter.AddPoint(s:String, x,y:Double) Method 204
.OptiRouter.Calculate(n:integer) Method 206
.OptiRouter.Clear() Method 206
.OptiRouter.ClearPoints() Method 206
.OptiRouter.DeletePoint(n:Integer) Property 206
.OptiRouter.ExecOptiRoute() Method 207
.OptiRouter.GetRouteFirst(n:integer):String Function 210
.OptiRouter.GetRouteNext():String Function 210
.OptiRouter.GetRouteClose() Function 211
.OptiRouter.CostPerGallon:Double Property 211
.OptiRouter.ExRadius:Double Property 212
.OptiRouter.GetFirstNode - Reserved Function 212

.OptiRouter.GetNextNode - Reserved Function 212
.OptiRouter.StopPoints[n]:PointRec 212
.OptiRouter.InsertPoint (n:Integer; pt:PointRec) Method 212
.OptiRouter.LoadFrom… Property 213
.OptiRouter.LoadPoints(s:String) Method 213
.OptiRouter.MemUsed:Integer Property 213
.OptiRouter.MovePoint(From, To: Integer) Property 213
OptiRouter.mpgCity:double Property 214
OptiRouter.mpgHwy:double Property 214
.OptiRouter.NumPoint(s:String) Property 214
.OptiRouter.PointExtent:TRPoint Property 214
.OptiRouter.Priority:RoadRec Property 215
.OptiRouter.RoutePath Interface 215
.OptiRouter.RouteParse(index:Integer; s:string):String Function 217
.OptiRouter.RouteType:TxRouteType Property 218
.OptiRouter.Speed:RoadRec Property 219
.OptiRouter.SaveToFile… Future 219
.OptiRouter.SavePoints(s:String) Method 219
.ImportLayer Interface 220
.Import.Brush Property 220
.Import.Count:Integer Property 221
.Import.Filename:String Property 222
.Import.Font Property 222
.Import.LabelField:Variant Property 223
.Import.LabelFilter:Integer Property 224
.Import.Lower Property 224
.Import.Mark Property 225
.Import.Name:String Property 226
.Import.Opacity:Integer Method 226
.Import.Pen Method 226
.Import.ProximityCheck:Boolean Method 228
.Import.ShowDialog Method 228
.Import.Upper Property 229
.Import.Visible:Boolean Property 230
.ImportManager Interface 231
.ImportMgr.AddLayer(Name:WideString, FileName:Widestring) Method 231
.ImportMgr.Clear Method 232
.ImportMgr.Count:Integer Property 232
.ImportMgr.CurrentIndex:Integer Property 232
.ImportMgr.CurrentLayer:ImportLayer Property 233
.ImportMgr.DeleteIndex(n:integer) Method 233
.ImportMgr.DeleteNamedLayer(Lname:string):Integer Method 234
.ImportMgr.Items[n]:ImportLayer Property 234
.ImportMgr.LoadfromFile(Fname:String) Method 234
.ImportMgr.SaveToFile(Fname:String) Method 235
.ImportMgr.ShowDialog Method 235
.ImportMgr.Visible:Boolean Property 236
.ImportMgr.ZoomLayers(N:Integer) Method 236
.POIManage Interface 237
.POIMgr.CATCount:Integer; Property 240
.POIMgr.CATVisibility(Index:OLEVariant):Boolean; Property 240

.POIMgr.Dialog; Method 241
.POIMgr.FormatHint:String Property 241
.POIMgr.FormatLabel:String Property 243
.POIMgr.GetCATCode(Name:WideString):Integer; Method 244
.POIMgr.GetCATName(CATcode:Integer):string; Method 244
.POIMgr.MilesMajor:Double Property 244
.POIMgr.MilesMinor:Double Property 244
.POIMgr.ReplaceBitmap(Index:OLEVariant; HBmp:Integer); Method 245
.POIMgr.Visible:Boolean Property 245
APPENDIX A - Overlay File Format 246
APPENDIX B – Color Palette 249
APPENDIX C – Configuration File 250
APPENDIX D – Abbreviations in Searching 257
APPENDIX E - Visual Basic Sample Code 258
APPENDIX F – CFCC Definitions 262
APPENDIX H – Underlay File Format 274
APPENDIX I – Autoload.Cty File Format 275
APPENDIX J – CMX File Format 276
APPENDIX K - Street Editing Mode 282
APPENDIX L - Delphi Code Examples 286
Printing to A printer Using DirectDraw 286
Saving to a Meta File 287
Using DirectView 288
APPENDIX M - Enumerated Properties & Record Structures 294
APPENDIX N - SAMPLE LICENSE AGREEMENT 297
APPENDIX O – DEPLOYING APPLICATIONS DEVELOPED WITH THE MapPro71.OCX, Release2 SDK 312

INTRODUCTION

MapOCX Pro (sometimes also referred to as MapPro) is a complete programming toolkit for incorporating detailed maps and spatial information in the Microsoft Windows environment. The self-contained ActiveX control was designed for quick and simple integration of high-quality, detailed map displays into any Windows visual environment, and facilitates brief development and deployment cycles. The latest version of the control is v7.1 (filename MapPro71.OCX).

This document is intended as an aid to developers, designers, and programmers in developing and integrating sophisticated map content into their applications. It is not meant for distribution to the end-users of applications based on this technology. It contains everything you need to use the MapOCX Pro toolkit. It also assumes you already are familiar with Microsoft's Visual Basic, and/or Borland's Delphi Integrated Development Environments (IDEs).

The sample code provided in this document is for demonstration purposes ONLY. As such, it is subject to modifications on a continuous basis as the MapOCX Pro evolves. The sample code does not constitute a software product, nor is it supported as such.

The MapOCX Pro 32-bit ActiveX control permits users to interface to an extensive vector database of roads, rivers, streams, ponds and other water bodies. The control has built-in behaviors which enable the user to dynamically zoom in or out, pan the map image, search for addresses, places, Zip Codes, Counties, perform Geocoding and Reverse Geocoding, and in general gives the developer/user the means of accessing vast amounts of geographical data.

The MapOCX Pro control has published properties, methods, events and dialogs. All parts of the control are written in Borland's Delphi.

Map Data

Two self-contained mapping databases may be licensed from Undertow Software, for use with MapPro OCX. One of these databases is based on the 2000 TIGER/Line data, from the Census Bureau, and the other is based on Premium Mapping Data supplied from TeleAtlas. Both data sets provide coverage for the Continental USA, Hawaii, Puerto Rico and US territories. The TeleAtlas data set also provides coverage for Canada.

Although both data sets provide resolution down to neighborhood roads, the TeleAtlas-based premium data set provides much more detailed address range coverage, additional resolution in the vector data, one way street information, exit ramp information, and in general a significantly higher degree of detail.

In addition to the detailed USA and Canadian data available with MapOCX Pro, a basic World reference data set is also provided, based on TeleAtlas, and enhanced with data from a number of other sources. This world data set, is meant to be used as a reference layer for users that may want to use their own underlay bitmaps, or overlays, for example, to produce maps for other parts of the world.

Stock Dialogs, Toolbars,…

Using just a single line of code, to access one of the numerous built-in stock dialogs and toolbars, provides the user with the ability to perform searches, print, perform street-level routing, set display options, and much more.

Map Rotation

On-the-fly map rotation, and a specialized "Heads-Up" mode allows the user to rotate map data to the preferred orientation, ideal for in-vehicle navigation.

Underlays & Overlays

Users can also load underlay bitmaps (satellite or aerial photography, BMP, JPG or GIF), and can create, load and edit overlays of CAD objects (lines, boxes, circles). Loaded overlay files may be of the older MapPro formats (.OVR or .CMX), of the new, advanced native CAD format, with support of attributable objects. A built-in CAD toolbar makes it easy for the developer to add vector elements that become part of the user CAD layer which is painted on top of the map surface.

The new CAD interface also allows the user to Group CAD objects in logical sets, modify individual or group object attributes, use built-in markers and wmf symbols to identify map locations, import wmf objects, an more. All these capabilities, are also available programmatically, for the developer that is interested in building their own CAD interface.

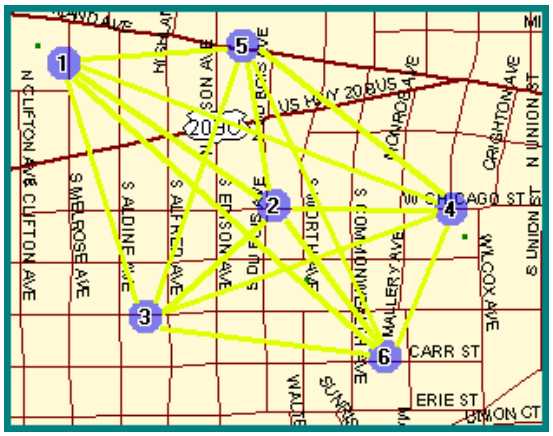
Geocoding and Reverse Geocoding

Although not designed as a dedicated geocoder, A variety of built-in search methods allow the user to Geocode based on an address, and the control also has built-in methods that permit the user to enter latitude and longitude coordinates and find the closest street, address, city, etc.

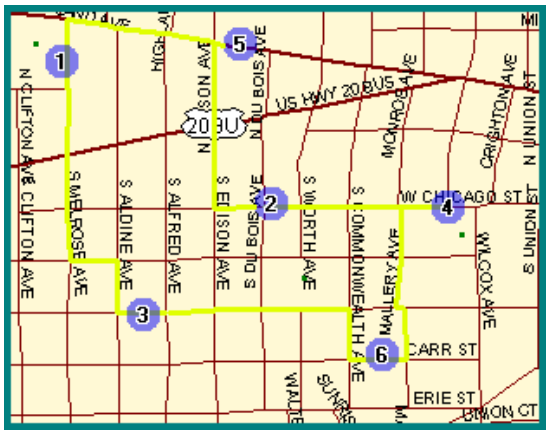
Route Optimization

A whole collection of new capabilities were added to the Map control in the new release, MapPro71. One of the most important additions is the Optimized Route Solution (ORS), many times referred to as the Traveling Salesman Problem (TSP).

This is the solution to the problem where a number of points need to be visited, and the user is interested in the optimized set of routes to all those points. In this implementation, the ORS can use either a closed circle path, i.e., a complete closed path is generated between **all** the user-specified points, or it can use fixed Start and Finish points. Although it may seem to be a trivial problem, for a couple of points, one can quickly see that the number of possible solutions that need to be examined, even for 6 pint, can quickly get very high.



Possible Routing Combinations for 6 points



ORS result for the same point

The ORS is implemented as a collection of Interfaces, Properties and Methods, described in the Optirouter section of this manual. below. Its capabilities can be selected both programmatically, or by invoking the Route Optimizer built-in dialog.

User Registration

The MapOCX control is registered in each development project, using the licensing information obtained from UnderTow Software, at the time of purchase, or (most likely) following the SDK's installation. No further interaction with Undertow is needed, by the developer. (See section on installing and using the control). **Please, note that the the Mappro SDK is distributed as a *Single User SDK*. If more than one developer will be using the SDK, on different computer system, a separate SDK license needs to be purchased for each such computer/user.**

All distributed, end-user copies of applications developed with MapOCX Pro, on the other hand, need to be registered with Undertow Software, individually, within 15 days of installation. The user is reminded upon startup of the application; they can register by phone or over the internet. This allows the developer to monitor the usage of their application and at the same time automates the licensing procedure.

What's Included in the Developers Toolkit

The MapOCX Pro Package contains the following:

- A choise of Map Data
  - 2000 TIGER/Line Street-level Data Set for USA, or
  - TeleAtlas Prime street-level Data for the USA (and /or Canada if licensed)
- Auxiliary files containing coarse world, state, county, place, hydro and roads data
- MapPro71.OCX ActiveX Component
- Developer's Users Manual
- REGSVR32.exe
- OC30.DLL
- Vendor Product Code and Developer Password (This is supplied to Licensee by phone)

Minimum Requirements

The following represent the minimum operating requirements for MapOCX Pro.

Minimum OS: **Operates On:**	Windows 2000 Windows 2000/XP
CPU	Pentium 4, 1 GHz+ or equivalent
RAM	256 MB
HD	100 MB (see note below)
CD-ROM	4x
Video	800x600, 256 colors

NOTE: 100 MB is assumed to be available in order to accommodate the OCX and associated files. The data is assumed to remain on the CD-ROM. If the data is transferred to the HD, then approximately 655 Mb (for Tiger Data) or 1+ Gb (for TeleAtlas) of space is required.

Coordinate System

The coordinate system used in the data accompanying the OCX, is a Lon/Lat (x,y), Cartesian coordinate non-projected map system. The system treats longitude and latitude as a simple rectangular coordinate system.

Scale, distance, area, and shape, in such a projection, are all distorted with the distortion increasing as one moves toward the poles.

Installing and Using the Control

The steps required to install and use MapPro71.OCX control are briefly described below:

1. Create a folder (subdirectory) and copy the MapPro71.OCX and associated files from the distribution disk to the folder. It is recommended that you retain the structure of the data directories (e.g., Data1, Data2, …) which makes it easier to troubleshoot problems should they arise later on.

2. Although some IDEs may transparently take care of this, it is recommended that you register the OCX with Windows. To do so, you may go to the command line, navigate to the folder where you installed MapPro71.OCX and REGSVR32.EXE, and

   Type: REGSVR32.EXE MAPPRO71.OCX

   And press ENTER.

3. Start your Development environment. If your IDE requires it (such as Delphi or Visual Basic, for example), then make sure you install the component in that environment. Refer to your IDE's documentation on installing components if necessary.

4. Create a new form and place the control on it.

5. Before you can create an application, you must enter in your Vendor Code (XXXX-XXXX) and your Password (XXXXXX). This information is either supplied when you purchase the MapPro71 SDK, or given to you by phone after you have received the SDK. In design mode, right mouse click on the MapOCX and select "Properties", then select the "Installation" tab where you can enter this information. If you are not yet a licensed developer or have lost your Developer ID, please contact Undertow Software at 1-978-794-9377 for further information. Note that this process, of entering Vendor Code and password, will need to be repeated every time you place a new instance of the control on a form in your project.

6. In the FormCreate event, or equivalent, set the Data Path properties so the OCX can find the data files (refer to Path_Data1, Path_Data2, Path_Data3, Path_States0, and Path_States1 in this manual for further information). Immediately following the path properties, also set the DataSoure property for the appropriate licensed set of data you will be using. It appears the loading sequence of some IDEs may interfere with loading sequence of the control. It is therefore recommended that in the FormCreate event, the developer also programmatically set the Vendor Code and Vendor Password.

Note: Once the OCX control is placed on a form, clicking the right mouse button, while the mouse cursor is on the control, will open the "Properties" dialog. If this "Properties" dialog is exited by clicking on the O.K. button, a new configuration file (MapPro71.CFG) containing the new options is saved to the disk. Furthermore, if any of the properties controlling the map view port are modified from the object inspector, the control is updated and a new MapPro71.CFG file is written to the disk.

Properties, Methods, Events

Only Properties, Methods and Events explicitly documented in this manual are supported and controlled by the OCX. Other inherited Properties, Methods and Events are available to the user through the control, depending on the development environment used. However, these are not controlled by the OCX and as such, support for them is beyond the scope of this document. There are also Reserved Methods that, although sometimes visible, are not supported for user access through the OCX.

Internal Data Structure

The data file structure of the map data, overlays, underlays, configuration files, and other support files used are proprietary. The format of some of these files is described in the appendices to this document.

Default Mouse Actions

Left Click (Re-centering)

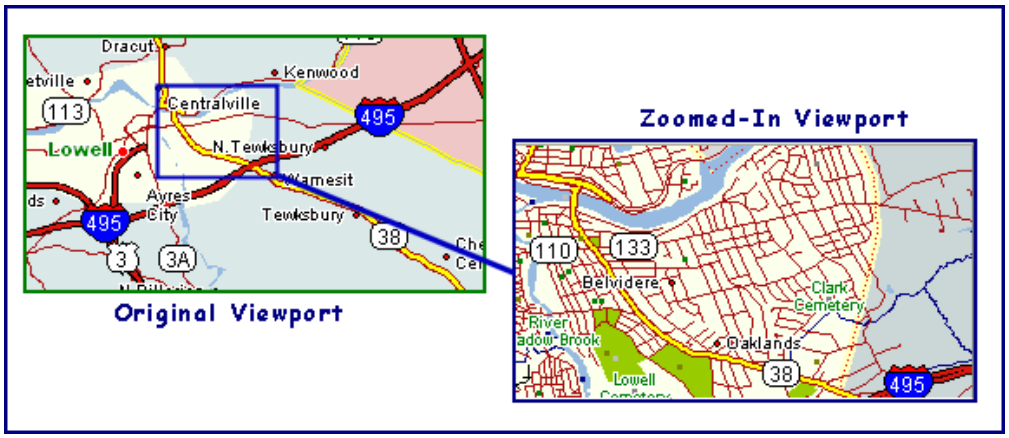
Pressing the Left mouse button, while in the default mode (mapMode=MdZoom), pans and repositions the map so that the clicked point is in the center of the ViewPort..

Right Click (Zooming Out)

Pressing the Right mouse button, while in the default mode (mapMode=MdZoom), zooms the map out by a factor of 2.

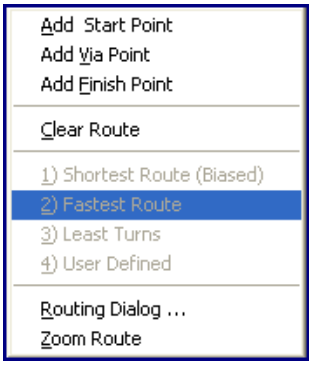
Left Click & Drag (Zooming In)

Pressing the Left mouse button, while in the default mode (mapMode=MdZoom), and dragging the mouse pointer, while holding the mouse button down, draws a dynamic rectangle on the screen. Once the mouse button is released, the map is zoomed in so that the area included in the selection polygon, fills the viewport.



Shift & Left Click (Quick Routing)

Pressing the Left mouse button, while holding down the Shift key, marks the current cursor location with a routing point marker, and pops up the Quick Routing menu *(Only enabled if RoutingActive is set to True – see info on the RoutingActive property).*

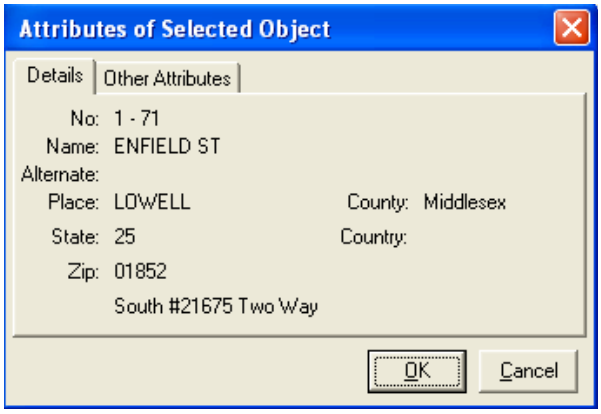


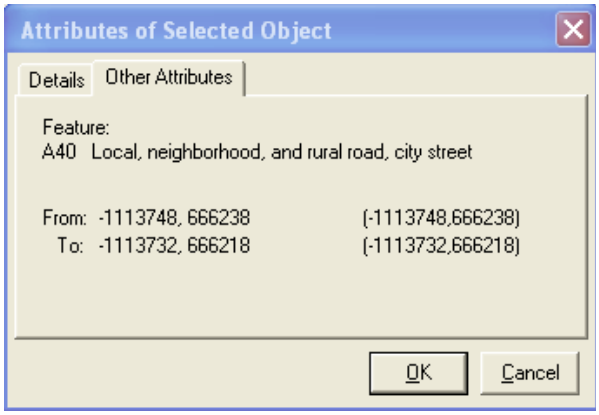
The quick routing menu allows you to add the Start, a Via or the End point for a routing calculation, Clear the currently calculated route, Zoom to the currently calculated route extents, open the main routing dialog, or perform a calculation using the currently defined points (the calculation options are disabled if at least two routing points have not been defined).

Note that points added this way are also accessible from within the OptiRouter dialog, as well.

Control & Left Click (Attributes)

Pressing the Left mouse button, while holding down the Control key, opens au a dialog displaying additional attribute data about the object at the clicked point. Note that this behavior is available only when the map is zoomed in to the point where local streets are visible, i.e., ~ 2 mile scale.





It should be pointed out that a number of other mouse-click behaviors are also available, when the map is not in its default mode, i.e., MapMode<>mdZoom, but those are described in the individual sections where the MapMode options are discussed.

Files in this package…

This section summarizes the various files distributed as part of the MapOCX Pro v7.1 developer's package with short description of their contents. Please, note that these files are subject to change without notice, as data gets modified/updated. For the latest files available at any given time, you may want to visit the Undertow Software web site, at www.undertowsoftware.com .

Root Directory of Distribution CD-ROM/DVD

MapPro71.OCX	The main mapping Active-X control.

Data1 Directory of Distribution CD-ROM/DVD

CRSHWY.ZPX	Coarse Hwy Network file, used by Optirouter
mAIR.ZPX	Airports points file
mCITY.ZPX	City/Place Points File
mCNTY.ZPX	Shaded County Polygons, below 2 mi scale
mCNTY03.ZPX	Shaded County Polygons, below 2 mi scale, Canada
mCNTYL.ZPX	Shaded County Polygons, above 2 mi scale
mHWY.ZPX	USA Interstate, State, and Local highways, used for rendering
mHYDRO.ZPX	Hydrographic features file
mPARK.ZP	Park polygons, above 2 miles.
mRTE.ZPX	USA Interstate and State Highways file (used for routing)
mRTE03.ZPX	Canada Interstate and State Highways file (used for routing and rendering)
mSHORE.ZPX	World Shoreline boundary, high definition
mSTATE.ZPX	State and County shaded polygons, above 2 mi scale
mWORLD.ZPX	High resolution shaded country polygons
VCRSHWY	Very Coarse Hiwhay Network (used by Optirouter)

Data2 Directory of Distribution CD-ROM/DVD

mLAND.ZPX	USA Landmarks file (used for rendering)
mLAND03.ZPX	Canada Landmark files (used for rendering)

Data3 Directory of Distribution CD-ROM/DVD

cities.BIN	Old, USA binary cities index file, used for searching
cities03.BIN	USA and Canada binary cities index file used for searching
fon.bin	Old USA binary phone area code index file, used for searching
place.ZPX	Additional Names Places Polygon file
zip.BIN	Old, USA binary ZiCode index file, used for searching
zip03.BIN	USA and Canada binary Zip/Postal Code index file used for searching

States Directory of Distribution CD-ROM/DVD

nnnnnnnn.ZPG	A series of state and Province files containing the high resolution Street level TeleAtlas Data (If the TeleAtlas data is licensed)
nnnnnnnn.ZP5	A series of state and Province files containing the high resolution Street level TIGER/Line based Data. (If Tiger Data is Licensed)
nnnnnnnn.SIG	A series of signature files required for end-user registration and use of any distributed applications. **Note:** End-user applications will not be operational if the appropriate *.SIG files are NOT presents in the states directorie(s). Also, the developer needs to be very careful about what files are present in that folder, because end-user registration pricing is keyed to

those files. You need to carefully read the "Deployment" section of this document.

Data4 Directory of Distribution CD-ROM/DVD (if POI data is Licensed)

POInn.CRA	POI Data in Proprietary format
POI0A.SIG	A signature file required for end-user registration and use of the POI data. **Note:** End-user applications will not be able to access the POI data if the appropriate *.SIG file is NOT present. The developer needs to be very careful about what files are present in that folder, because end-user registration pricing is keyed to those files. You need to carefully read the "Deployment" section of this document (Appendix O)

Underlays Directory of Distribution CD-ROM/DVD

*	Contains a number of underlay files (vary from release-to-release)

Sample Source Code Directory of Distribution CD-ROM/DVD

*	Contains sample Source code the demonstrates the use of the ActiveX control in a number of different development environments, in different archives. Each archive is named according to the development environment it is for, and contains all the necessary files for the user to compile and run a sample project.

Development Considerations – Quick How To's and things to be aware of

Rotated Maps
When working with rotated maps, screen updates may be slightly slower due to the additional vector manipulations required for the rotation.

Shading
When using DirectDraw, the shading of places and MCD areas is based on a dynamic, "First-Come-First-Served" basis. Two DirectDraw processes on two different surfaces on the screen could result in different shades for the same place or MCD area.

Sample Code
Any samples of code provided in this manual are for Delphi or Visual Basic, and are not part of any integral application. Each example is meant solely as an illustration of the procedure or property for which it is given and not necessarily in combination with the other code snippets in this documentation.

Searching for Places & Geocoding
There are a number of different methods that may be used for searching. The most versatile method is using **GeoFind**, primarily for visual environments, or the **GeoFindFirst/Next** and **GeoFindArray** for non-visual environment. The GeoFind collection of methods attempts to perform a fuzzy search based on a partial street address string specified by the user. Other more specific, explicit, methods are the **FindCity, FindZip and FindStr**. The results from these method-based searches can be parsed and used to locate a Lat,Lon set of coordinates for a specific address location.

If a visual interface is of interest, then a collection of dialog based, specific searches are available through the **ExecSearch** tabbed dialog interface. There are also the **ExecStreet, ExecPlace** and **ExecZipCode** search dialogs, which use older technology and older index files. They are not recommended and were only left in the newer releases of the OCX for compatibility with older applications.

Reverse Geocoding
One of the most powerful features of this ActiveX control, is it's ability to Reverse Geocode, i.e. find the street address or place name closest to a specified set of coordinates. This is achieved through the use of the **FindClosest** and **FindClosestPlace** methods.

Displaying Data Points
A number of methods are available for the user to display their own data points on the map. A built-in marker can be displayed at a given Lon,Lat, by using the **DrawObject** method (if called in the OnPaintAfter event, then it will be painted at those coordinates every time the screen updates). A built-in symbol can be places at the desired coordinates using one of the methods in the .CAD interface (**.CAD.Marker**, **.CAD.MetaObject**, etc.). Large sets of user specified points may also be displayed through the **ExecDbImport** method, which plays the items on the map, therefore not using precious memory resources for rendering. Finally, maximum flexibility is provided to the user by being able to use and manage a set of user items, that can be placed at any Lon,Lat location, accompanied by a user-specified bit-map, an identifying string, etc. This is achieved though a collection of methods, **SetItem**, **SetItemString**, **SetItemBitmap**, **SetItem Angle**, etc.

Adding your own streets
Although the streets databases are updated on a on-going basis, given the tens of millions of road segments, and the continuous additions of new housing developments, new roads, etc., and the time necessary for such information to be collected by the Census Bureau (or our prime data suppliers),

verified, processed, etc., it's possible that you may find localized road issues that you need to address, by temporarily augmenting the built-in database. The MapOCX Pro control enables you to do that by setting **MapMode=2**, and then using the steps described in Appendix –K.

Single Instantiation – Many Map Views

The system is set up for a single MapPro control instance in the user's application. Users are warned not to use a second control, as they tow controls would share some of the data segments, thus corrupting each other. Instead, they should use either the **DirectDraw**, or better yet, the **DirectView** methods to create additional map views in a single application. Note that in some IDEs, the user may also have to issue a FormX.Show, followed by a FormX.Hide, prior to using a FormX.ShowModal when using more than one form.

Distribution Files

When distributing your application to end-users, you need to include all files in the States, Data1, Data2, and Data3 directories, in the distribution disk(s) you received from Undertow Software. Files in other directories may be optional, depending on the feature set your application is using.

It is strongly recommended that **Regsvr32.exe** and **oc30.dll** should be distributed with your application's setup and installed in the Windows System directory. Although most operating systems contain these files, some do not. It is also recommended that you install the MapPro71.OCX in the application directory when distributing your application, and if possible, automate the Windows OCX registration process through your installation procedure.

*.SIG Files

The data sets distributed with this package, contain a number of files with the extension **SIG** in the states folder. These files **need** to be there for the street level data to be accessible. These SIG files need to also be present in the street-level data folder, in the any distribution application package to the end user. Note, however, that if you are not using Canadian data in your distributed application, then you should NOT include the TANACAN08.SIG file in your distribution package. If you do, then the end-user registration process will automatically charge your vendor account for the Canadian data use, for every end-user that registers with Undertow Software Corp. even though your application may not need/use it.

Upgrading from an earlier Version

- If you are upgrading from an earlier version of the control, you should be aware that the MapPro71.OCX uses a new CLSID and can therefore co-exist with prior versions of the control, installed on the same system. This does not apply to MapPro71.OCX Release 1 and Release 2 that have the same CLSID, and they look as the same control, to the Operating System.
- If you are replacing an instance of an older version of the control in one of your projects, with this one, you should note that depending on your IDE, you may have to re-link all events in your project. Note that in most IDEs, in order to have a new control interface registered, the IDE has to be shut down first, as it will not permit a new OCX file to replace an existing one, and/or to be registered, if an instance of the control is currently being used.
- If you are replacing an old build of MapPro71.OCX, with a more recent one, it is recommended that you re-import the interface, to make sure that any new interface additions to the control are properly registered.

Street Level Data Visibility Problems

One of the most common problems developers encounter, is that the OCX appears to work as expected, only when they zoom in, there is no street level data visible. 99% of such problems reported, it turns out to be a data path issue. If you experience such problems, before you contact technical support, please make sure that you check the data settings, by echoing the respective OCX properties at the point in your code where you'd

expect the street-level data to be visible. Echo the properties Path_States, Path_Data1, Path_Data2, Path_Data3, Data4 and DataSource, and make sure they are what you'd expect them to be at the point. Do not assume that because you set them somewhere in code, they must be so. Different IDE timing considerations in autoloading the control's configuration files, or default directories at the time of execution, may be affection what the above properties are pointing to. Also, make sure that the appropriate *.SIG files are present in the states foldr.

Drawing on the Maps

Although use of the built-in .CAD interface is straight forward, and does not require the developer to manage the CAD objects, there may be instances when the developer needs to draw their own lines, on the map. This can be achieved be using the built-in **DrawLine** method. Note that if you want to give the line(s) a "sticky", behavior, i.e., be drawn every time the map is redrawn, then any such user-controlled drawing should be done in the **OnPaintAfter** event. The additional benefit is that the event also makes the map's dc available to the user, so they do not have to get and release dc's. The user may also use the various Windows APIs (**MoveToEx, LineTo**, etc.) to draw on the surface of the map, either in the OnPaintAfter event (since the dc is available), or anywhere in their application by using the Windows **GetDC** and **ReleaseDC** APIs.

Routing

Because of the difference pricing in licensing applications with or without routing, the default setting is for the MapPro71. Rel 2 OCX not to permit routing calculations. These capabilities can be activated by setting the RoutingActive property of the OCX to True.

Setting of this property also results in the appropriate end-user license cost being charged at the time of registration of the client license (see Appendix N and Appendix O for more details).

World Map Extents

The default behavior of the OCX is to restrict the viewport to North America. In order to have the ability to zoom, pan, etc. outside the North America extents, the "World Extents" flag needs to be set to true by calling the **SetOption with Opcode= $C004** (See SetOption section).

MapOCX Pro Licensing

The MapOCX Pro SDK is only licensed for development purposes. Before you distribute an application which contains MapOCX Pro or use the control in a server environment, you need to obtain a licensing and distribution Agreement from Undertow Software. A "sample" license agreement is included in this document. Because this sample may not be up to date, call 978-794-9377 for the latest version of the Agreement, and for further information and pricing of the end-user (client) licenses.

Technical Support

In the event you have a technical question about the MapOCX, support is available at support@undertowsoftware.com, www.undertowsoftware.com or call (978) 794-9377.

When you call, you should be at your computer, have the appropriate product documentation, and prepared with the following information:

- The product and version you are using
- The hardware and operating system you are using
- The exact wording of any messages that appeared
- A description of what happened and what you were doing when the problem occurred

Updates

It is recommended you frequent our web site, www.undertowsoftware.com, where maintenance releases of the MAPOCX71.OCX will be placed from time to time.

OCX Reference Section

AboutBox() — Procedure

Displays a dialog showing the OCX name, version and copyright information. This information is required to be displayed in the about box of any application using the OCX.

VB Example

```
Private Sub Command1_Click()
  ‘ Open the About Box
  MapPro1.AboutBox
End Sub
```

Delphi Example

```
Procedure TForm1.Button3Click(Sender: TObject);
begin
 {Display the OCX name, version, etc.}
 MapPro1.AboutBox;
end;
```

AddViaPoint(x,y:Double; s:String) — Procedure

Adds the specified point to the list of Via points that are used for routing. Note that Via points can be added, and managed, from within the Routing dialog, as well.

X,Y - Lon/Lat coordinates (decimal degrees)
S - Identifying string

VB Example

```
Private Sub Command84_Click()
  ‘ Add three points to the Via points array
  MapPro1.AddViaPoint -89.4, 43, "MyViaPoint1"
  MapPro1.AddViaPoint -112.4, 42.12, "MyViaPoint2"
  MapPro1.AddViaPoint -118.23, 35.55, "MyViaPoint3"
End Sub
```

AutoConfig:Boolean — Property

When this property is FALSE, the configuration file is saved only when the Config procedure is called. If it is TRUE, the configuration file is saved automatically when a property is changed, and when the application is exited. The default config file name used by the control is MapPro71.cfg The configuration file may also be loaded by the user, at will, using the LoadConfig property.

Note: Depending on the development environment, the sequence of instantiation and initialization of the OCX, and loading the Config file may be different. If during your development testing, it appears that Properties being set in your code, at run time, appear to have different values once the

control is created, then set AutoConfig to False and repeat your testing. It could be that the loading of a config file happens *after* initialization of such properties.

VB Example

```
Private Sub Command1_Click()
  MapPro1.AutoConfig = true
End Sub
```

Delphi Example

```
Procedure TForm1.Button3Click(Sender: TObject);
begin
  MapPro1.AutoConfig:=true;
end;
```

AutoPaint:Boolean — Property

When this property is FALSE, the surface of the OCX control is NOT painted unless a Redraw or Repaint command is issued. The value of this property may be set when designing the application, or at run time. If the value of this property is true, then all zoom operations and any other operation that is deemed to have changed any of the viewport properties results in automatically repainting the map.

Normally this is turned off by the developer if they know that a lot of operations affecting the viewport are going to take place - in order to avoid time-consuming map repaints, and then set to True again at the end to finally update the map.

VB Example

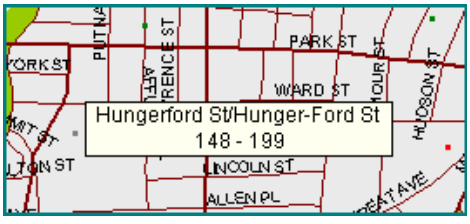
```
Private Sub Command1_Click()
   ' force a repaint of the map
  MapPro1.AutoPaint = true
End Sub
```

Delphi Example

```
Procedure TForm1.Button3Click(Sender: TObject);
begin
  MapPro1.AutoPaint := False;
  MapPro1.Miles := 0.03;
  MapPro1.GotoPoint(-88.5,41.56);
  MapPro1.AutoPaint:=true;
  MapPro1.Redraw;  //Map changes scale & goes to point in 1 step
end;
```

AutoQuery:Boolean — Property

When this property is TRUE, visible features with a name are described by a floating hint window, when the cursor is placed on them. In addition if the feature is a road or street with an address range, this information will appear below the name. The value of this property may be set when designing the application, or at run time.



VB Example
```
Private Sub Command1_Click()
   MapPro1.AutoQuery = false
End Sub
```

Delphi Example
```
Procedure TForm1.Button5Click(Sender: TObject);
begin
  MapPro1.AutoQuery:=false;
end;
```

BevelInner:Integer — Property

Sets the state of the inside bevel of the OCX control and is used to create 3D effects. The value of this property may be set from the properties editor, when designing the application, or at run time.

Valid states are
0 - BvNone
1 - BvLowered
2 - BvRaised

VB Example
```
Private Sub Command1_Click()
  MapPro1.BevelInner = 1
  (or MapPro1.BevelInner = BvLowered  could be used)
End Sub
```

Delphi Example
```
Procedure TForm1.Button5Click(Sender: TObject);
begin
  MapPro1.BevelInner:=1;
  { or MapPro1.BevelInner:=BvLowered; could be used }
end;
```

BevelOuter:Integer — Property

Sets the state of the outside bevel of the OCX control and is used to create 3D effects. The value of this property may be set from the properties editor when designing the application, or at run time.

Valid states are
0 - BvNone
1 - BvLowered
2 - BvRaised

VB Example
```
Private Sub Command1_Click()
  MapPro1.BevelOuter = 0
  (or MapPro1.BevelOuter = BvNone  could be used)
End Sub
```

Delphi Example
```
Procedure TForm1.Button6Click(Sender: TObject);
begin
  MapPro1.BevelOuter:=0;
  { or MapPro1.BevelInner:=BvNone; could be used }
end;
```

BevelWidth:Integer — Property

Sets the width of the inner and outer bevels of the OCX control, in pixels.

VB Example

```
Private Sub Command1_Click()
  MapPro1.BevelWidth = 2
End Sub
```

Delphi Example

```
Procedure TForm1.Button7Click(Sender: TObject);
Begin
  MAPPRO1.bevelwidth:=4;
End;
```

BorderWidth:Integer — Property

Sets the width of the border around the OCX control, in pixels.

VB Example

```
Private Sub Command1_Click()
  MapPro1.BorderWidth = 2
End Sub
```

Delphi Example

```
Procedure TForm1.Button3Click(Sender: TObject);
begin
  MapPro1.Borderwidth:=2;
end;
```

CAD:Interface — Interface

Allows the user access to the CAD interface, in order to draw CAD objects on the map surface. See later sections in this document for a detailed description of this interface.

VB Example

```
Private Sub Form_Activate()
  MapPro1.InitNonVis
  QrMode = 0
  ‘ Display the CAD toolbar mode
  MapPro1.Cad.Toolbar.Mode = TbFix
  List1.AddItem MapPro1.PostUnderlay
End Sub
```

CenterView(X,Y:Integer) — Procedure

Takes the x,y cursor position, in device coordinates relative to the OCX, calculates the Lon/Lat equivalent to that position and centers the map about the point with the calculated coordinates, at the current scale factor.

Note: The GotoPoint Procedure can be used to locate the viewport using Lon/Lat coordinates instead.

VB Example

```
Private Sub Command1_Click()
  MapPro1.CenterView(20,20) 'Center the viewport at screen
  'coordinates 20,20 from top left corner.
End Sub
```

Delphi

```
Procedure TForm1.Button7Click(Sender: TObject);
begin
  {Center the viewport at screen coordinates 20,20 from the
  top left corner of the control}
  MapPro1.CenterView(20,20);
end;
```

ClearExclusion() — Procedure

Clears the current exclusion boundaries from memory. These boundaries are used to exclude enclosed road segments from routing calculations. See the SetExclusion, SaveExclusion and LoadExclusion methods as well, for more information.

Delphi Example

```
Procedure TForm1.Button34Click(Sender: TObject);
// Clear the current exclusion list
begin
  MapPro1.ClearExclusion;
  ExcludeRt:=false;
end;
```

ClearStreets() — Procedure

Clears memory of currently loaded User streets file. (see SaveStreets, LoadStreets). Also, see Appendix-K for details on the Street-Editing mode, and how to create and load these User street files, which can be used to augment supplied mapping data for new streets added since the generation of the datasets..

VB Example

```
Private Sub Command2_Click()
    MapPro1.ClearStreets
End Sub
```

ClearOverlay() — Procedure

Clears memory of the currently loaded User Overlay file. (See OpenOverlay). Also, see Appendix-A for details on the format of the overlay file. It only applies to the older type OVR/CMX overlays.

VB Example

```
Private Sub Command2_Click()
    MapPro1.ClearOverlay
End Sub
```

ClearViaPoints() Procedure

Clears All currently defined Via (routing) points, from the Via points list. It also clears the point markers and the highlighted route from the map surface. Also see the AddViaPoints and the routing dialog descriptions about adding via points to a routing calculation.

ClrStrBubble() Procedure

Clears the bubble that appears on the screen following a Street search operation using the ExecStreet procedure. This is needed because the bubble object intentionally remains on the screen once it has been created through the ExecStreet call. This way, a street "searched for" by the user can remain identified, as the user pans the map around.

VB Example

```
Private Sub Command1_Click()
  MapPro1.ClrStrBubble
  MapPro1.Redraw
End Sub
```

Delphi Example

```
Procedure TForm1.Button7Click(Sender: TObject);
begin
  {Execute the ExecStreet procedure and locate a street}
  MapPro1.ExecStreet;
  { After the ExecStreet procedure has been executed and
  the street has been located, eliminate the bubble }
  MapPro1.ClrStrBubble;
end;
```

Color:Integer (or enumerated ColorName) Property

Inherited Property. Color of the background, i.e., the color for Mexico and Canada. The default color is yellow.

VB Example

```
Private Sub Command1_Click()
  MapPro1.Color = vbGreen
End Sub
```

Delphi Example

```
Procedure TForm1.Button1Click(Sender: TObject);
begin
  { Set Background color to System Green}
  MapPro1.color:=clGreen;
end;
```

Coords:Integer Property

Sets the coordinate notation in the ExecSearch and ExecLonLat dialogs as well as the echo area in the toolbar.

If Coord is set to: 0 (or 'CdLonLat')

The **Lon,Lat** convention is used, consistent with the rest of the OCX.

1 (or 'CdLatLon')
The ExecSearch and ExecLonLat dialogs and the coordinates echoed in the toolbar area use the **Lat,Lon** form.

VB Example

```
Private Sub Command1_Click()
  MapPro1.Coords = 1
End Sub
```

Delphi Example

```
Procedure TForm1.Button21Click(Sender: TObject);
begin
  { Set Coordinate format to lat/Lon}
  MapPro1.Coords:=1;
end;
```

Cursor:Integer — Property

Cursor type. Meaningful only when the MapMode property is set to MdUser. A positive value will use one of the pre-defined cursor types in the OCX. If no cursor type is defined for that value, then a stock Windows pointer cursor will be returned. Zero or a negative value will use one for the Windows stock cursor types. "-1" results in a non-visible cursor. However, if the "edgepan" property is true, even if Cursor=-1, the standard MdZoom OCX cursor will be selected.

Although cursor types can be selected using this property, it is recommended that the 'Custom' property is used to select the cursor type, instead, as it gives more control to the user. It should be pointed out that if the value of Cursor is set to 2, i.e., pointing to the second position in the cursor array, and mapMode is set to mdUser, then the cursor defined by .Custom is used.

Note: The cursor types defined in the OCX are subject to change without notice. It is recommended that either Windows stock cursors or user-created ones are used.

VB Example

```
Private Sub Command12_Click()
  CurId = 105
  MapPro1.MapMode = MdUser
  ' Load Cursor from Resource File
  Picture = LoadResPicture(CurId, vbResCursor)
  MapPro1.Custom = Picture.Handle
  MapPro1.Cursor=2
End Sub
```

Delphi Example

```
Procedure TForm1.Button4Click(Sender: TObject);
begin
  {* Set User mode and the left arrow cursor
  defined in the OCX *}
  MapPro1.mapmode:=MdUser;
  MapPro1.cursor:=17;
end;

procedure TForm1.Button3Click(Sender: TObject);
```

```
// Another example of using a custom cursor.
var h1,custCursorNo:integer;
begin
  custCursorNo:=strtoint(edit2.Text);
  if custCursorNo in[1..2] then
 begin
   MapPro1.MapMode:=mdUser;
   if custCursorNo=1 then h1:=loadcursor(0,IDC_NO)
       else h1:=loadcursor(0,idc_APPSTARTING);
   MapPro1.Custom:=h1;
 end else MapPro1.MapMode:=mdZoom;
end;
```

Custom:Integer — Property

It sets the handle to the Cursor type, when the MapMode property is set to MdUser. If this property is undefined, when MdUser is set, the cursor type defaults to the Windows CR_Default type. The handle used for setting this property may be obtained by the Windows API calls, CreateCursor, or LoadCursor.

Notes: This property may be set at any time, and not just prior to setting MapMode to MdUser. It will remain set and become active when MapMode = MdUser. It should also be noted that the Custom cursor definition is placed in position "2" of the Cursors array, so Cursor has to be set equal to "2" for Custom to work.

VB Example

```
Private Sub Command1_Click()
  MapPro1.Mapmode=mduser
  h1=loadcursor(0,idc_wait) ' any valid "*.ani" or "*.cur"
  MapPro1.custom=h1
  MapPro1.cursor=2
End Sub
```

Delphi

```
procedure TForm1.Button3Click(Sender: TObject);
     var h1,custCursorNo:integer;
     begin
       custCursorNo:=strtoint(edit2.Text);
       if custCursorNo in[1..2] then
       begin
         MapPro1.MapMode:=mdUser;
         if custCursorNo=1 then h1:=loadcursor(0,IDC_NO)
             else h1:=loadcursor(0,idc_APPSTARTING);
         MapPro1.Custom:=h1;
         MapPro1.Cursor:=2;
       end else MapPro1.MapMode:=mdZoom;
 end;
```

DataSource:txDataSource — Property

Enumerated variable specifying the dataset to use used. The available choices are shown below. Please not that in some IDEs, the enumerated values are not accessible and the integer values (from the left column shown below) need to be used.

#	Enumeration	Data Set to Use	Files
0	Z_NONE	No Data	None
1	ZP5_TIGER	Tiger USA Data Set	*.ZP5
2	ZPG_GDTUSA	TeleAtlas USA Data Set	*.ZPG
3	ZPG_RESV1	Reserved	None
4	ZPG_CDN	TeleAtlas CANADAData Set	*.ZPG
5	ZPG_RESV2	Reserved	None
6	ZPG_GDTUSACDN	TeleAtlas USA and CANADA Data Set	*.ZPG

It should be noted that each data set contains files in different format, and they are not interchangeable. Also note that a signature file needs to be present in the directory with the state files.

It is recommended that this property is set in your code, not just in the property inspector to ensure consistent data access.

Delphi Example

```
procedure TForm1.Button37Click(Sender: TObject);
// Toggle use of datasets
begin
  if MapPro1.DataSource=ZP5_TIGER then
  begin
     MapPro1.DataSource:=ZPG_GDT;
     Button37.caption:='Data:GDT';
     Label3.Caption:='Using: GDT, Set Data Paths...';
     button38.enabled:=false;
     button11.click;
  end else
  begin
    MapPro1.DataSource:=ZP5_TIGER;
    Button37.caption:='Data:TIG';
    Label3.Caption:='Using: TIGER, Set Data Paths...';
    button38.enabled:=true;
    button11.click;
  end;
end;
```

DbFile:String — Property

Sets the database file (dbf format) to be used for displaying pints on the screen. (Also see the ExecDbImport method).

DegFormat:TxDegFormat — Property

Sets the format for the string labeling Grid lines displayed on the map.

0 or dfDEC	=	Decimal degrees (dd.fffffff)
1 or dfDECMIN	=	Degrees, decimal minutes (dd mm.fffffff)

2 or df DMS = Degrees, minutes, seconds (dd mm ss)

VB Example

```
Private Sub Command80_Click()
  If i=0 then MapPro1.DegFormat = dfDEC
  MapPro1.refresh
End Sub
```

DeleteAllitems() Procedure

Clears all objects from the Object layer (the user layer where all user created bitmap items are placed). It also clears the cache used to store user bitmaps. Note that deleting a single user item does not release the cache space used by it.

Note: See the SetItem and related procedures for information on how to place user created bitmaps on the control surface, or other surfaces in the user's application.

VB Example

```
Private Sub Command1_Click()
  MapPro1.DeleteAllItems
End Sub
```

Delphi Example

```
Procedure TForm1.Button8Click(Sender: TObject);
begin
  {Delete all user created bitmaps from the displayed map}
  MapPro1.DeleteAllItems;
end;
```

DeleteItem(ID:Integer) Procedure

Deletes object with a given ID from the object layer.

Note: See the SetItem and related procedures for information on how to place user-created bitmaps on the control surface, or other surfaces in the user's application.

VB Example

```
Private Sub Command1_Click()
  Call MapPro1.DeleteItem(5)
  'Or
  MapPro1.DeleteItem 5
End Sub
```

Delphi Example

```
Procedure TForm1.Button11Click(Sender: TObject);
begin
  {* Deletes the fifth item created by the user *}
  MapPro1.DeleteItem(5);
end;
```

DeleteItemRange(ID1,ID2:Integer) — Procedure

Deletes all user created items from id1 to id2. Id1 must be less than id2. If id1 is greater or equal to id2, no action takes place.

Note: See the SetItem and related procedures for information on how to place user created bitmaps on the control surface, or other surfaces in the user's application.

VB Example

```
Private Sub Command1_Click()
  Call MapPro1.DeleteAllItemRange(3,11)
  Or
  MapPro1.DeleteItem 3,11
End Sub
```

Delphi Example

```
Procedure TForm1.Button3Click(Sender: TObject);
begin
  {Deletes items 3 to 11, inclusive, created by the user}
  MapPro1.DeleteAllItemRange(3,11);
end;
```

DeleteViaPoint(ID:Integer) — Procedure

Deletes specified Via point from the routing list. Also see the AddViaPoint method, and the Routing dialog for adding/deleting Via points interactively.

VB Example

```
Private Sub Command101_Click()
  ' Delete Via point #1  (note 0-based list)
  MapPro1.DeleteViaPoint 1
End Sub
```

DevCode:String — Property

The value is assigned to each unique customers product. This is used in conjunction with DevPass to unlock the OCX for development. This property is only set within a non-visual environment. In a visual environment, you would normally enter the developer code and password by right mouse clicking on the control and selecting the properties menu item and then the installation tab. (Also see ExecRegister, DevPass)

Note: Contact Undertow Software for Developer Codes and Passwords (978) 794-9377.

VB Example

```
Private Sub Form_Load()
'Used to develope with the OCX is a non-visual environment
 MapPro1.DevCode = "1234-1234"
 MapPro1.DevPass = "134534"
 MapPro1.ExecRegister (5432)
End Sub
```

DevPass:String — Property

DevPass is a unique password to use the DevCode. This is used in conjunction with DevCode to unlock the OCX for development. This property is only set within a non-visual environment. In a visual environment, you would normally enter the developer code and password by right mouse clicking on the control and selecting the properties menu item and then the installation tab.

Note: Contact Undertow Software for Developer Codes and Passwords (978) 794-9377.

VB Example

```
Private Sub Form_Load()
'Used to develope with the OCX is a non-visual environment
 MapPro1.DevCode = "1234-1234"
 MapPro1.DevPass = "134534"
 MapPro1.ExecRegister (5432)
End Sub
```

DirectDraw(dc, offX, offY, Wd, Ht : Integer, Clear, Scale, BW : Boolean) — Procedure

Draws the map onto the windows display context (DC) provided by the user. When the DirectDraw procedure finishes painting, it triggers an OnDirect event, prior to resetting the scale that was used to map the OCX control size to the user specified DC size (if scale was set to true). This allows the user to use Windows API calls to draw to the same output device specified by the DC, without having to worry about scaling problems.

The DirectDraw process works as follows:

- Determines scale needed to map the OCX to the DC
- Clears the background (if specified by the user)
- Triggers the OnDirectBefore event
- Paints the Map
- Paints the overlay
- Triggers an OnDirect event
- Paints the user layer
- Resets scale

dc: Device context

offX, offY: The X and Y offsets from the top left corner of the control.

Wd, Ht: The width and height of the output in device units.

Clear: A boolean flag that instructs the OCX to clear the control background prior to drawing on it. This is very useful when sending output to a printer device (generally, Clear should be true). It might be set to false if the user wants to paint his own background on the control, prior to issuing the DirectDraw command. Note that the current viewport will be adjusted to fit the specified output dimensions.

Scale: A boolean flag (True/False), which indicates whether the DirectDraw output should be scaled or not. It is recommended that when printing, this flag should be TRUE. Note, however, that when this is TRUE, the quality of the printout depends on the size of the OCX control surface. The smaller the OCX surface, the larger the scaling that needs to be applied, creating the possibility of jagged, coarse lines.

BW: A boolean flag (True/False) that defines if the printout is monochrome (TRUE), or color (FALSE).

Notes: When using the DirectDraw procedure, it is up to the user to determine whether the print device supports direct bitmap scaling, which affects the print quality of user-created objects. This could be done through the Windows API call GetDeviceCaps (consult your Windows documentation for more details). DirectDraw uses the BitBlt API, however, when the scale flag is set to true, Windows internally uses the StretchBlt API.

It should also be noted that the user should make certain the color mapping mode of a B&W print device is correctly set when printing color bitmaps. It is recommended that monochrome bitmaps be used when attempting to print. If a color bitmap is used, the color of the lower left corner pixel of the bitmap is assumed to be the background color (white), and every other color maps to black.

The Windows API in DirectDraw uses a variable of type TRect to identify the size and position of the scaleable polygons used for highway shield bitmaps and other bitmaps placed on the control surface. It assumes that the input coordinates are in device units - not Twips. The user must ensure that the mapping mode for the control is set to mmText. Furthermore, when operating under Visual Basic, the user must ensure that the parent form of the control has its map mode set to device units.

When creating large bitmaps using Directdraw, the user should be conscious of the resources required. For example, using DirectDraw to create a bitmap to be eventually be printed on a high resolution color device could cause the system to run out of resources. Mono bitmaps are recommended in such situations. See Appendix-L for sample code on how to use DirectDraw for printing.

VB Example

```
Private Sub Command1_Click()
  MapPro1.directdraw(dc,0,0,Image1.Picture.bitmap.width,
              Image1.Picture.bitmap.height,true,true,true)
  MapPro1.redraw
End Sub
```

Delphi Example

```
Procedure TForm1.Button10Click(Sender: TObject);
{ Simply draws the map directly to an image control. Note
  that in this simple example, the width and height in
  device units do not need to be calculated, they are
  obtained from the image control properties }
var dc:hdc;
begin
   dc:=Image1.Picture.bitmap.canvas.handle;
   MapPro1.directdraw(dc,0,0,Image1.Picture.bitmap.width,
                         Image1.Picture.bitmap.height,true,
                         true,true);
   MapPro1.redraw;
end;
```

Distance(x1,y1,x2,y2: Variant):integer **Function**

Calculates the distance between two points given their Lon/Lat coordinates. The calculated distance is in thousandths of a mile, using a modified Great Circle distance formula, optimized for speed.

x1,y1: Lat, Lon coordinates of first point

x2,y2: Lat, Lon coordinates of second point

VB Example

```
Private Sub Command1_Click()
  MapPro1.Distance(x1,y1,x2,y2)
End Sub
```

Delphi Example

```
Procedure TForm1.Button3Click(Sender: TObject);
var ax,bx,ay,by:real;
begin
  ax:=-123;
  bx:=-121.35;
  ay:=43.756;
  by:=35.1927;
  with MapPro1 do
  begin
    Panel1.caption:='Distance: '+str(Distance(ax,ay,bx,by)*1000);
  end;
end;
```

DirectView(dc, w, h:integer; var LonL, LatB LonR, LatT: Double; var DvScale:Double)

Draws a map using the current viewport settings, on a control surface specified by the user. Ideal for situations where more than one map need to be visible, at different scales and viewport extents. (See Appendix-L for a sample Delphi project using DirectView.

Where: **dc** – handle of the control surface onto which the DirectView map will be drawn

W, H – The width and height of the control Surface

LonL,LatB,LonR,LatT – the longitude and latitude coordinates of the desired map.

DvScale – Option that specifies the criteria to be used for determining the Viewport extents.

(1) If the DvScale parameter is zero, then the method sets the DirectView viewport extents to those passed by the user, modifying them for any needed aspect ratio adjustments, and returns the adjusted extents and the calculated MapPro1.miles property as DvScale.

(2) If the DvScale parameter is NOT zero, then the method uses the extents passed by the user to calculate a DirectView viewport centerpoint, sets the mappPro61.miles parameter to the value of DvScale and returns the calculated extents of the DirectView window.

Delphi Example

```
Procedure Tform1.DoWindow1(image:timage;f:double);
// Use Picture box for DirectView image
var dc:integer; bmp:tbitmap; x1,y1,x2,y2:double;
sf:double;
begin
  // Create default bitmap if not present
  if image.picture.bitmap.width=0 then
  begin
```

```
    bmp:=tbitmap.create;
    bmp.width:=image1.width;
    bmp.height:=image1.height;
    image.picture.bitmap:=bmp;
  end;
  // The handle to the bitmap obtained this way is transient,
  // Lock it to ensure it remains usable by DirectView
  image.picture.bitmap.canvas.lock;
  dc:=image.picture.bitmap.canvas.handle;
   with MapPro1 do
   begin
          x1:=lonleft;
    x2:=lonright;
          y1:=latbottom;
    y2:=lattop;
   end;
   messagebeep(0);
   sf:=MapPro1.miles /f;
   label6.caption:=floattostr(x1);

MapPro1.DirectView(dc,image1.width,image1.height,x1,y1,x2,y2,sf);
   label8.caption:=floattostr(x1);
   image1.picture.bitmap.canvas.unlock;
   image1.Refresh;
end;
```

DMS(x: OleVariant): string — Function

Converts a double value from decimal degrees to a DEG:MIN:SEC String (also see See LLMode, Coords)

VB Example

```
Private Sub Command1_Click()
  Text1.Text = "Lat: " + CStr(MapPro1.DMS(MapPro1.Ycord))
         + ", Lon: " +   CStr(MapPro1.DMS(MapPro1.Xcord))
End Sub
```

Delphi Example

```
Procedure TForm1.Button9Click(Sender: TObject);
begin
  with MapPro1 do
  begin
    ax:=xcord;
    ay:=ycord;
  end;
  {Set Panel2 to display the coordinates given by xcord,
   ycord, in Deg:Min:Sec format}
  Panel3.caption:=DMS(ax)+' , '+DMS(ay);
end;
```

DrawBubble(dc:integer; x,y:integer; S:string) — Procedure

Draws a text bubble at the user specified Lat/Lon (x,y)coordinates, containing specified text. Using a CR (CHR(12), will act as a line break and a carriage return.

Note: The object created by DrawBubble is NOT part of the map; it is temporarily drawn on the screen. If the user wanted to retain the object, for subsequent drawings, provisions would have to be made for storing it and repainting it on the screen, or the user should draw it in the OnPaintAfter event handler.

VB Example

```
Private Sub Command1_Click()
  Call MapPro1.DrawBubble(Dc,-125,45, "Sample" &  Chr(13) &
"Break")
  'Or
  MapPro1.DrawBubble Dc,-125,45, "Sample" & Chr(13) & "Break"
End Sub
```

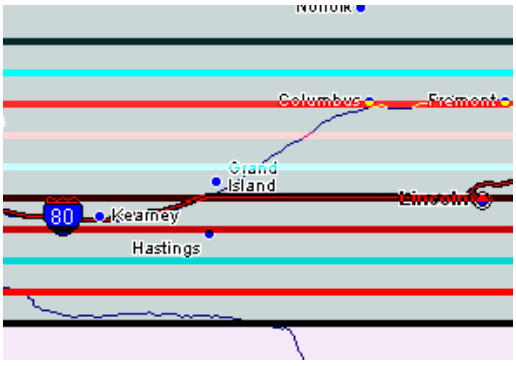
Delphi Example

```
Procedure TForm1.MapPro1paintAfter(Sender: TObject;dc: Integer);
begin
  {Draw a text bubble on top of the map surface,
   at -125,45 Lon/lat coordinates }
  MapPro1.Drawbubble(dc,-125,45,'Sample');
end;
```

DrawLine(dc:integer; x1,y1,x2,y2:OleVariant;w,cl,mode: Longint) Procedure

Draws a line between two points using the dc and the parameters specified by the user and described below. The object created by DrawBubble is NOT part of the map; it is temporarily drawn on the screen. If the user wanted to retain the object, for subsequent drawings, provisions would have to be made for storing it and repainting it on the screen, or the user should draw it in the OnPaintAfter event handler.

x1,y1	=	Coordinates of point #1 in Longitude and Latitude.
x2,y2	=	Coordinates of point #2 in Longitude and Latitude.
w	=	Width of the line in pixels.
cl	=	RGB color of the brush
mode	=	Any of the supported Windows raster operations, e.g. R2_CopyPen, R2_MAskPen, R2_Black,… The graphic below show the result of a number of Raster Operations with a Red color line. From top to bottom they are: R2_BLACK, R2_COPYPEN, R2_MASKNOTPEN, R2_MASKPEN, R2_MASKPENNOT, R2_MERGENOTPEN, R2_MERGEPEN, R2_MERGEPENNOT R2_NOTCOPYPEN, R2_NOTMERGEPEN.



Note: The drawn line is clipped against the edges of the drawing window.

VB Example

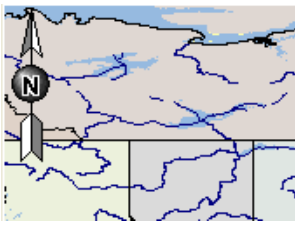
```
Private Sub Command1_Click()
  MapPro1.DrawLine(dc,-87.65,41.84,-118.24,34.05,W,Color,Mode)
End Sub
```

Delphi Example

```
Procedure TForm1.Button18Click(Sender: TObject);
{------------------------------------------------------}
{ Draw a line using the control's DrawLine Method}
{------------------------------------------------------}
var dc,w,color,mode:longint;
begin
  {get the dc for the map object}
  dc:=getdc(MapPro1.handle);
  {set the color to blue}
  Color:=clblue;
  {set the mode to raster operation to R2_MergePen}
  Mode:=R2_MergePen;
  {set the width to 10 pixels}
  W:=10;
  {Draw a line from Chicago to LA.   Note that if the screen}
  {is updated, the line is not redrawn unless it's tied to}
  {the OnPaintAfter event}
  MapPro1.DrawLine(dc,-87.65,41.84,-118.24,34.05,W,Color,Mode);
  {finally, release the dc}
  releasedc(handle,dc);
end;
```

DrawNorth(dc,x,y:integer) — Procedure

Draws an arrow indicating the North direction, using the device context and the screen coordinates (X,Y) specified by the user. The X,Y=0,0 point is the top left corner of the dc, consistent with the Windows convention.



VB Example

```
Private Sub Command38_Click()
  MapPro1.DrawNorth GetDC(MapPro1.Handle), 15, 45
End Sub
```

Delphi Example

```
Procedure TForm1.Button8Click(Sender: TObject);
var dc:longint;
begin
  {Draw a North Arrow after painting the map on the dc }
  dc:=getdc(MapPro1.handle);
  MapPro1.DrawNorth(dc,15,45);
  releasedc(dc,handle);
end;
```

DrawObject(dc:integer,Lon,Lat:OleVariant;Item,Color: Longint) Procedure

Draws the predefined icon specified by Item, and the specified color, at the Longitude and Latitude coordinates specified by Lon & Lat, on the control specified by the device context, dc. A white highlight is drawn around the item as well.

The predefined items are identified below:

1 - Circle
2 - Square
3 - Diamond
4 - Triangle
5 - Left arrow
6 - Right arrow
7 - Up arrow
8 - Down arrow

VB Example

```
Private Sub Command1_Click()
  MapPro1.drawobject (GetDC(MapPro1.Handle), tempYcord,
  tempXcord, 1, vbGreen)
End Sub
```

Delphi Example

```
Procedure TForm1.Button18Click(Sender: TObject);
{--------------------------------------}
{ Draw an Object at the cursor location }
{--------------------------------------}
begin
  dc:=getdc(MapPro1.handle);
  DrawObject(dc,MapPro1.xcord,MapPro1.ycord,2,clRed);
  releasedc(handle,dc);
end;
```

DrawScalebar(dc, x, y: Integer) Procedure

Draws a scale bar at the specified x,y screen (logical) coordinates. The user must provide a Device Context (DC). For example this procedure could be used to place a scale bar on the left side of the bottom status bar similar to that displayed in Precision Mapping Streets. (Also see the ScaleBar property for an alternative way of displaying a scale bar on the map, without using a DC)

VB Example

```
Private Sub Command41_Click()
  MapPro1.DrawScaleBar GetDC(MapPro1.Handle), 5, 8
End Sub
```

DelphiExample

```
Procedure TForm1.MapPro1paintAfter(Sender: TObject;dc: Integer);
begin
  { Since this is part of the OnPaintAfter event, which is
    part of the OCX control, there is no need to get the
    device context. It's just 'dc'.  The scale bar is drawn
    with its top left corner starting at 5 units horizontally
    and 8 units vertically from the top left corner of the map
```

```
    control }
  MapPro1.drawscalebar(dc,5,8);
end;

Example:
Procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  { Here, the procedure paints using a paintbox canvas
    handle, rather than getting a dc }
  MapPro1.DrawScaleBar(paintbox1.canvas.handle,8,2);
end;
```

EdgePan:Boolean — Property

Boolean property which when TRUE allows fixed edge pan behavior to operate. When active, this mode changes the cursor appearance when within 8 pixels from the frame edges. When the left mouse button is pressed, the viewport pans in the indicated direction.

VB Example

```
Private Sub Command39_Click()
  MapPro1.EdgePan = True
End Sub
```

Delphi Example

```
Procedure TForm1.Button2Click(Sender: TObject);
begin
  MapPro1.EdgePan:=True;
end;
```

EdgePanAmount:Double — Property

The fraction of the viewport to advance during an EdgePan operation. For example, if EdgePan=0.5, then a pan operation will result in 50% of the viewport being panned.

VB Example

```
Private Sub Command39_Click()

' Set EdgePan to 25% of viewport
  MapPro1.EdgePanAmount = 0.25
End Sub
```

EdgePanWidth:Integer — Property

The number of the pixels from the edge, defining the EdgePanZone. When the cursor is within EdgePanWidth pixels from the edge, and EdgePan is turned on, the cursor changes to the Edgepan mode and clicking it pans the map in the indicated direction.

VB Example

```
Private Sub Command92_Click()
  MapPro1.EdgePanAmount = 0.5
  MapPro1.EdgePanWidth = Val(Text26.Text)
```

```
  MapPro1.EdgePan = Not (MapPro1.EdgePan)
End Subb
```

Enabled:Boolean — Property

Inherited property determines whether the OCX control will be able to receive mouse and keyboard messages.

VB Example

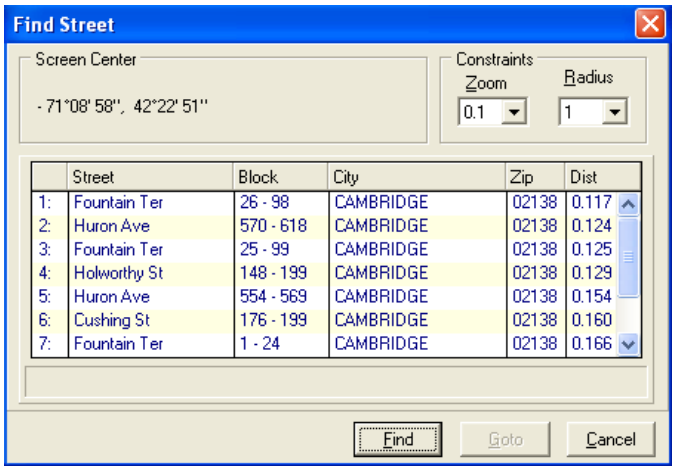
```
Private Sub Command39_Click()
  MapPro1.Enabled = True
End Sub
```

Delphi Example

```
Procedure TForm1.Button2Click(Sender: TObject);
begin
  MapPro1.Enable:=True;
end;
```

ExecClosest() — Procedure

Presents the user with a dialog that permits the user to search and display the 10 street segments closest to the current view port center point (its coordinates displayed in the dialog). Double clicking on one of the 10 names will reposition the viewport around that point, at the Zoom scale specified by the user.



VB Example

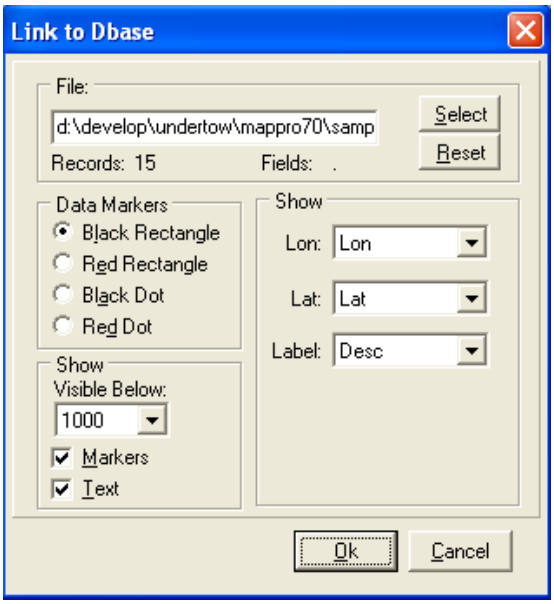
```
Private Sub Command39_Click()
  MapPro1.ExecClosest
End Sub
```

Delphi Example

```
Procedure TForm1.Street3Click(Sender: TObject);
begin
  MapPro1.ExecClosest;
End;
```

ExecDbImport() Procedure

Presents the user with a dialog that permits them to interface to a dbf database file, and display points from the file on the map. When the specified file is opened, it is searched for fields named LON and LAT, and if it finds them it assumes that they contain the corresponding coordinates for each record in the database (in decimal degrees). It also assumes that the field immediately following the LON and LAT fields is text and will be used for labeling the points. Note that the user can override thes fields and has control over displaying a label, or not, as shown below. (Also see the dfFile property).



- The user may click Select to navigate to the desired dbf file.
- The fields in the "Show" panel are populated with the autodetected fields to be used for Lat, Lon and labeling, but the user may select from any of the available fields in the drop down list.
- The user may select the type of marker to use, select the upper visibility scale (in miles), and select whether to display a marker, the label, and/or both.

ExecLonLat() Procedure

Presents user with a dialog that permits the input of Latitude and longitude coordinates, and upon confirmation, places the viewport around the point specified by the user.

VB Example

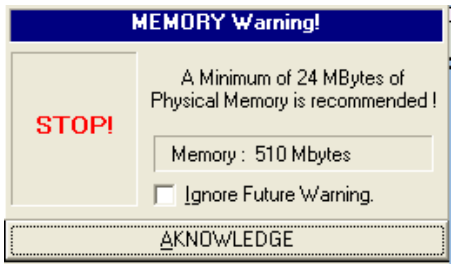
```
Private Sub Command39_Click()
  MapPro1.ExecLonLat
End Sub
```

Delphi Example

```
Procedure TForm1.Street3Click(Sender: TObject);
begin
  MapPro1.ExecLonLat;
end;
```

ExecMem(X:LongInteger):LongInteger — Function

Returns the amount of RAM available on the system, in bytes. Furthermore, if the amount of memory detected is less than the number specified in the function call, then the OCX will display a message warning box indicating that the amount of memory available may be not enough to execute the application effectively. For example, here is the error message dialog that would appear with the call ExecMem(505000000) on 512 Mb system.



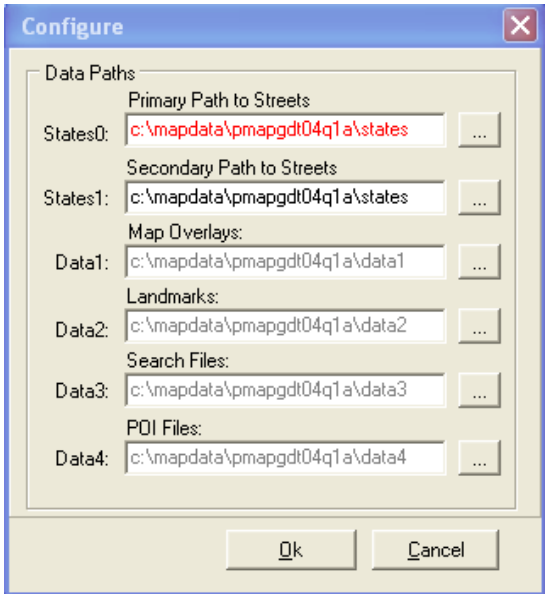
Delphi Example

```
procedure TForm1.Button5Click(Sender: TObject);
var i,j:LongInt;
begin
  //  Check to see that at least 92 MB RAM is found
  i:=92*1024*1024;
  j:=PMAP21.execMem(i)
end;
```

ExecMethod(s:String) — Procedure

Invokes dialogs based on the standard dialogs of Precision Mapping Streets 4.0.

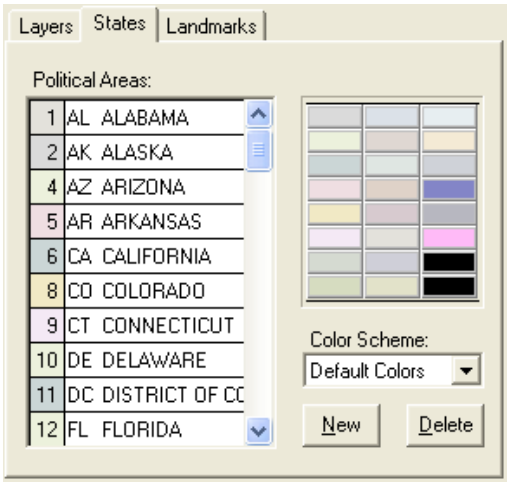
1. **s = CONFIG** Invokes the File, Config dialog which permits the user to set the data paths either by simply typing them in, or by navigating through their system (navigate by clicking on the ellipses next to each data path)



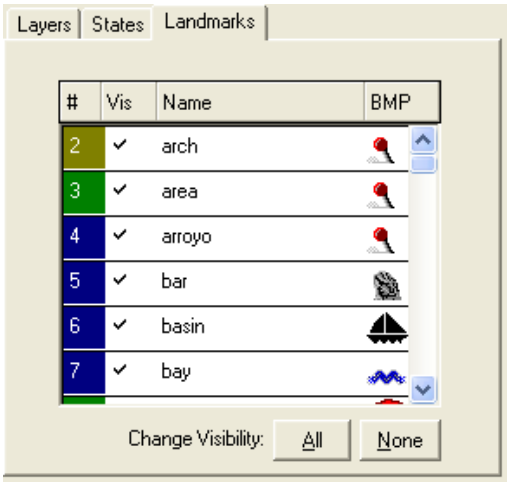
2. **s = LAYER** Invokes the Options, Screen_Options dialog which is made up of three different tabs as described below.

- **Layers Tab**

- **States Tab**

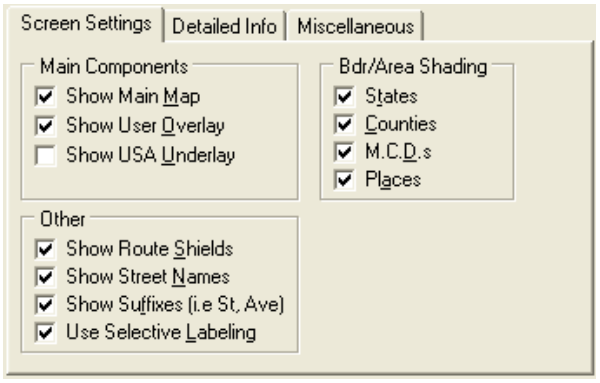


- **Landmarks Tab**

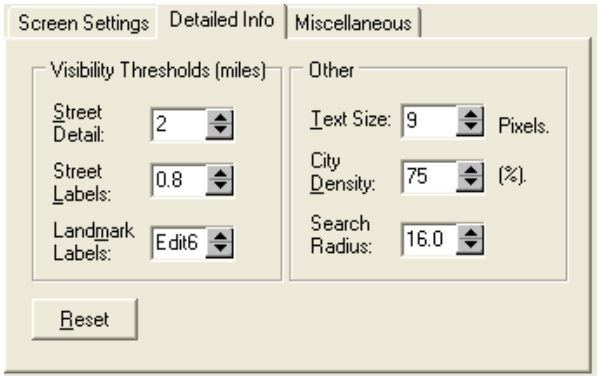


3. **s = DISPLAY** Invokes the Diaply Options dialog. This dialog is comprised of 3 tabs as shown and explained below:

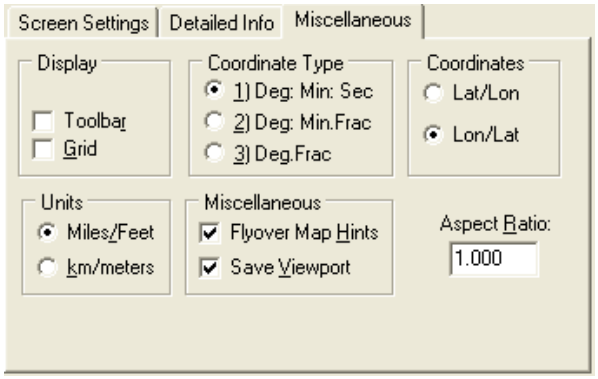
- **Screen Settings**



- **Detailed Info**



- **Miscellaneous**



VB Example

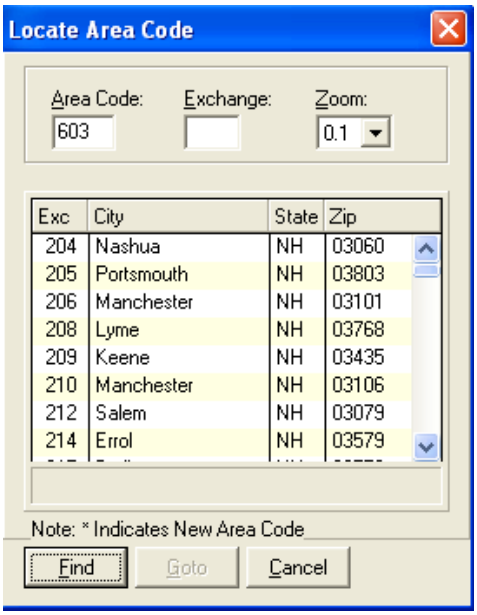
```
Private Sub Command_Click()
  MapPro1.ExecMethod("COnfig")
End Sub
```

Delphi Example

```
procedure TForm1.Button5Click(Sender: TObject);
Begin
  MapPro1.ExecMethod('DISPLAY');
End;
```

ExecPhone() — Procedure

Presents user with the standard Precision Mapping Streets phone search dialog. Once a AreaCode is specified and the search is completed, the user is presented with a listbox containing the retrieved matches. Selecting one of the listbox choices presented, by double clicking, will center the viewport at the Lon/Lat of the selected AreaCode centroid.



VB Example

```
Private Sub Command39_Click()
  MapPro1.ExecPhone
End Sub
```

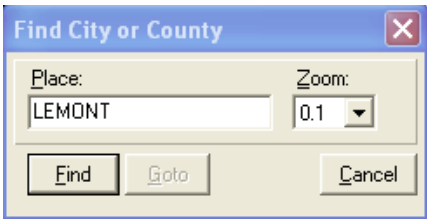
Delphi Example

```
Procedure TForm1.Button1Click(Sender: TObject);
begin
  MapPro1.execPhone;
end;
```

ExecPlace() — Procedure

Presents user with the standard Precision Mapping Streets place search dialog. Once a Place is specified and the search is completed, the user is presented with a listbox containing the retrieved matches.

Selecting one of the listbox choices presented, by double clicking, will center the viewport at the Lon/Lat of the selected Place centroid.



Entering a place name followed by an asterisk (wildcard) will find all places that contain the specificed strint, otherwise an exact match search is performed.

VB Example

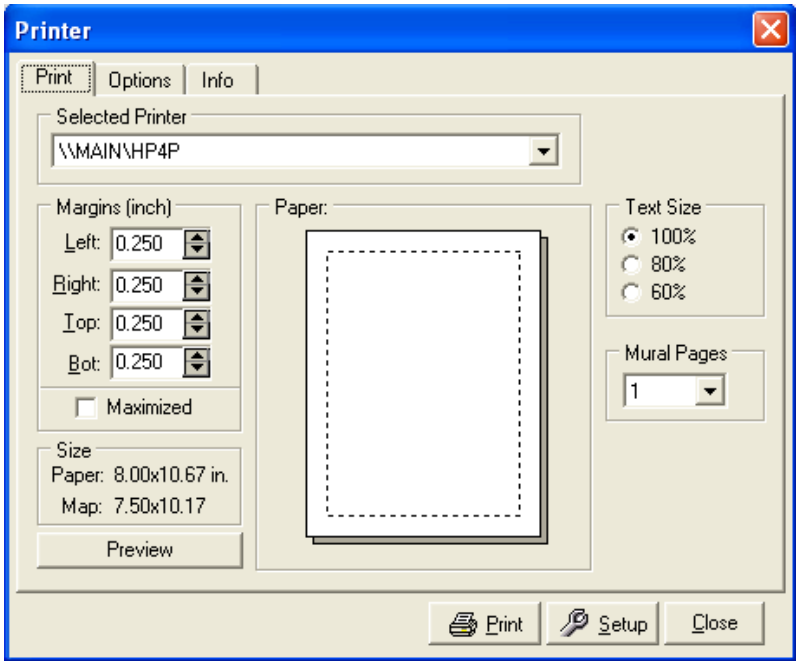
```
Private Sub Command39_Click()
  MapPro1.ExecPlace
End Sub
```

Delphi Example

```
Procedure TForm1.Place1Click(Sender: TObject);
begin
  MapPro1.execplace;
end;
```

ExecPrint() **Procedure**

Presents user with a Print dialog, where the print device, margins, etc. may be specified. Note that for more control over the print process, the DirectDraw method may be used with a printer device as the output. Once the Print dialog opens, the following options are available to the user, arranged in 3 tabs.



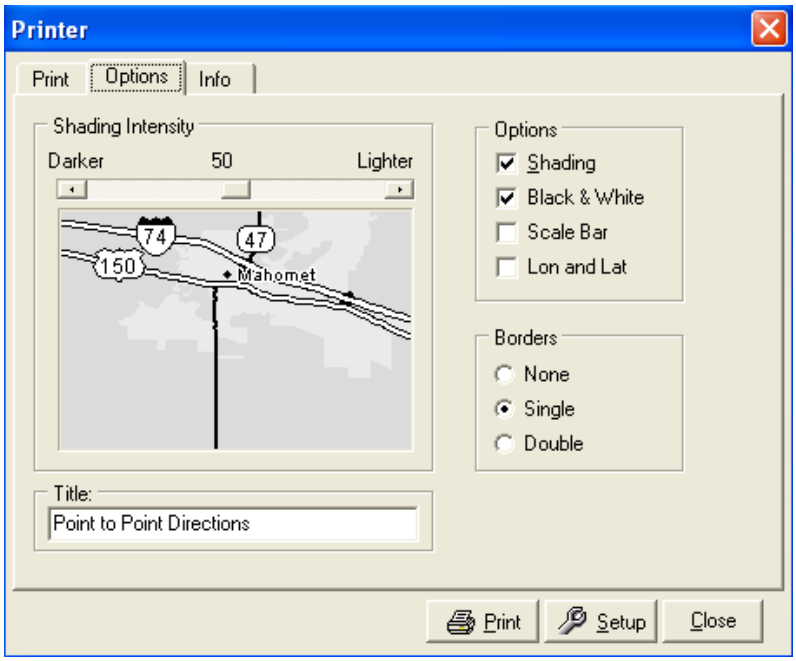
The first tab (Print) allows the user to…

- Set the page margins. (clicking on maximize sets the margins to correspond to the maximum printable area for the currently selected print device)
- Select the Text size, as a percent of the default size.
- Preview the selected map area that would be printed
- Select the number of murals to use (up to 16). When selecting more than one mural, the control will print a key map showing the location of each mural grid, followed by the mural grids themselves. Note that when printing multiple tiles, the viewport is resized to retain the scale, but account for the different aspect ration of the control on the screen and the paper.

Print — Prints the map using the current settings and the currently selected print device.

Setup — Opens up the windows printer setup dialog

Close — Returns control back to MapPro71.OCX.

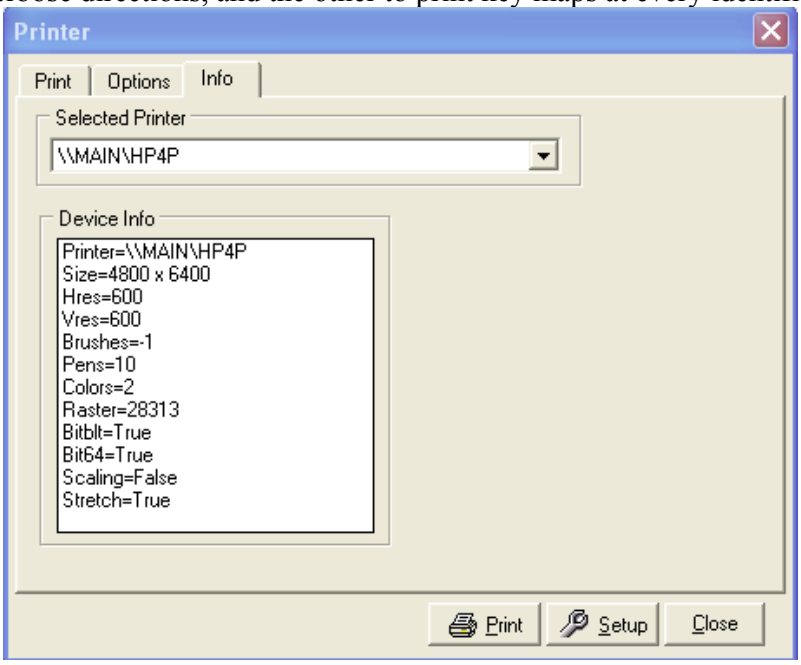


The Options tab permits the user to:

- Set Shading intensity (this would affect the shading or parks, water bodies, and in general all filled polygons). As you change the Shade Intensity values, the Preview area reflects the current intensity of the image to be printed.
- Select whether to use Shading.
- Select whether to print color or B&W
- Select whether to print Lon/Lat lines, or not.

- Select whether to print the scale bar
- Select what type of borders to use
- Set the title to be printed on top of each printout of the map.

If the print dialog is invoked from the Routing dialog, then there are two extra options. One to print the Routing detailed, verbose directions, and the other to print key maps at every identified turn of the route.



The Printer tab contains detailed information regarding the currently selected printer device, which is obtained by directly querying the device. This information can be very helpful in identifying and solving printing problems.

ExecPrintEX() — Procedure

ExecPrintEX() opens the same dialog as ExcPrint, with the addition of having the options available when printing from within one of the routing dialogs.

ExecRegister(x:LongInt) — Procedure

ExecRegister(5432) will present the user with the registration dialog box if the MapOCX is not registered. This will allow them to enter the registration code if needed. The Registration process is managed by the OCX, so there should normally *not* be a reason for the developer to call this routine, but it may be useful in some circumstances. Contact Undertow Software Corp. for information regarding self-registering end-user applications.

VB Example

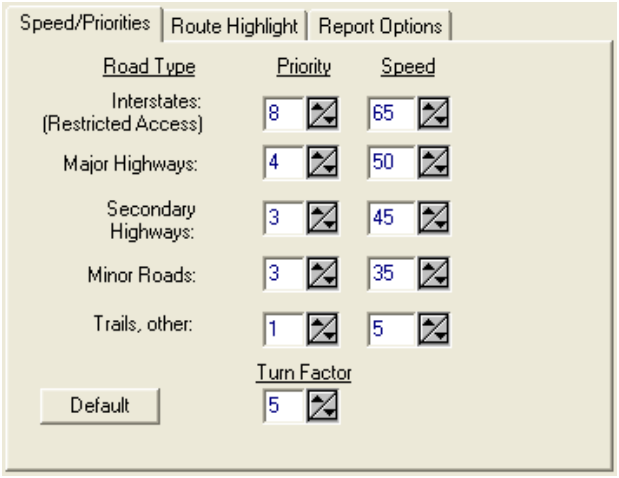
```
Private Sub Form_Load()
 'Opens the registration dialog
 MapPro1.ExecRegister (1)
 MapPro1.ExecRegister (5432)
End Sub
```

ExecRoadOption() — Procedure

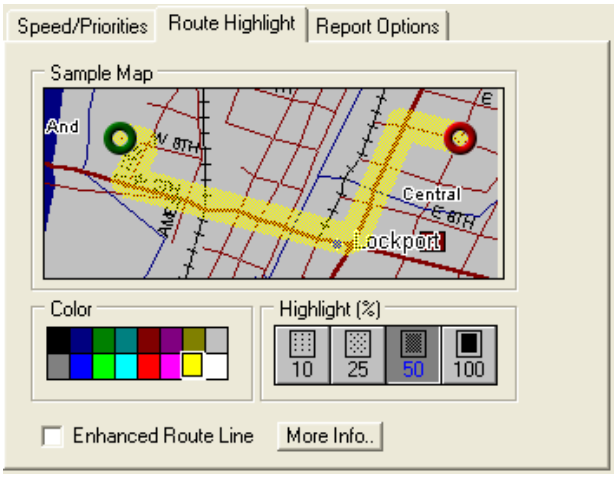
Allows the user to specify advanced options for calculating the optimum route between two points. Also see FindRoute, ExecRoute.

The options are organized in a number of tabs, as described below.

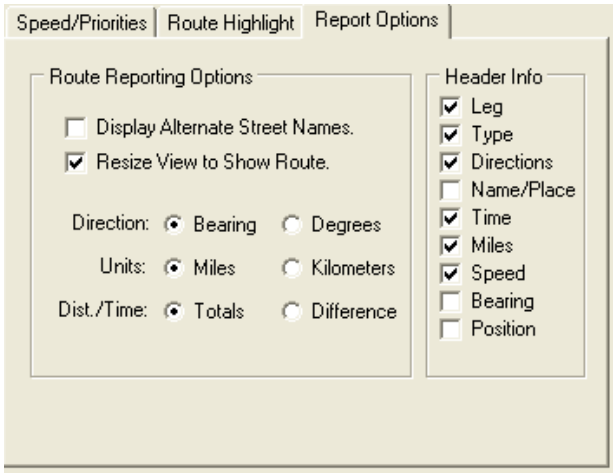
Speeds: In this dialog, the user may set the travel speeds (mph) for 6 different types of roads from Limited Access (Interstate) Highways, to Trails, etc.



Priorities: The user is permitted to set the travel preference priority for the 6 types of roads identified in the "Speeds" tab. A value of "1" sets the lowest priority while a value of "5" sets the highest. A value of "0" excludes that road type from routing calculations.



Other: In this dialog, the user is permitted to select the type of Route highlighting on the map: Dot(10%), Dot(25%), Dot(50%), Cross, Hatch, and Grid, as well as the color of the route highlight.



Notes: These are just the options to be used when doing routing calculations. The actual route calculation is done using either the FindRoute, or the ExecRoute methods. The user should be cautioned that the likelihood of routing failure increases when road types are totally excluded, i.e., when travel preference priority is set to "0" for one or more road types.

VB Example

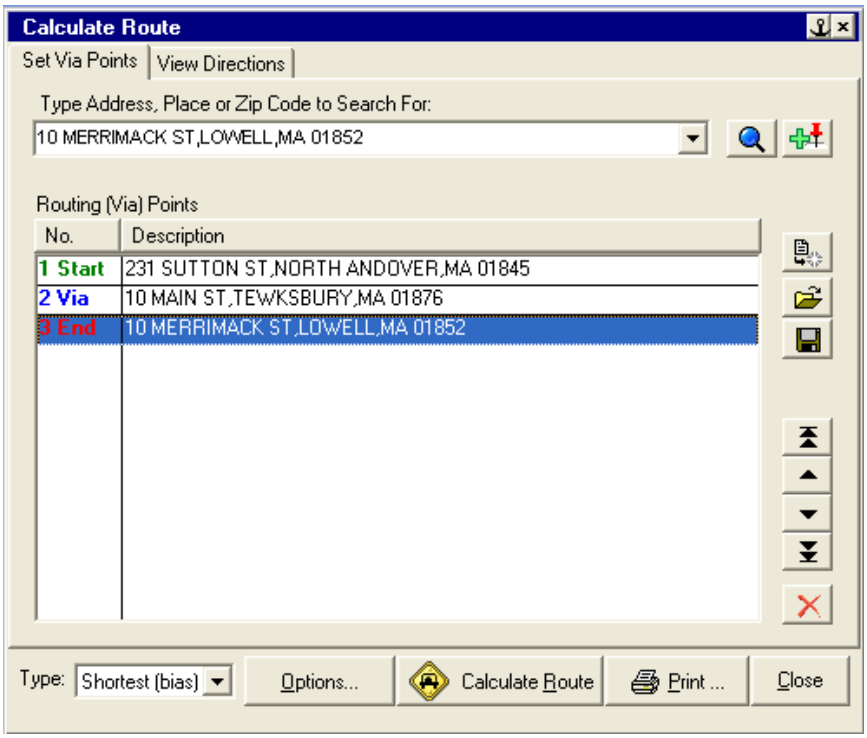
```
Private Sub Command39_Click()
  MapPro1.ExecRoadOption
End Sub
```

Delphi

```
Procedure TForm1.Button6Click(Sender: TObject);
{------------------------------------------------------}
{ Invoke the Options Dialog                            }
{------------------------------------------------------}
begin
  MapPro1.ExecRoadOption;
end;
```

ExecRoute(x1, x2, x3, x4: Double) **Procedure**

Presents user with a dialog that permits the input necessary to perform a Routing calculation. They X1, x2, x3, x4 parameters are remnants from an earliee implementation and were left for backward compatibility. They can be set to *any* value and have no effect. This routing dialog may also be invoked by clicking on the routing icon of standard toolbar. *(Only enabled if RoutingActive is set to True – see info on the RoutingActive property).*



When the dialog is first opened, the focus is on the Set Via Points tab, which allows the user to perform the following operations.

Search for the address or location entered by the user in the edit field of the combo box. If an exact match is found, then the located address is entered in the edit field. If multiple addresses (hits) are found, then the combo box opens up listing all of them. The user needs to select one of the returned hits, at which time the selected hit (address) is entered in the edit field of the combo box.

Add the address in the edit field to the routing list. Note that this button is disabled until a search has been performed and a valid address or location has been returned and selected by the user. If this is the first point being added, it becomes the Starting point for the routing calculation. If it's the second point being selected, then it becomes the End (finish) point for the routing calculation. If it's the 3rd, 4th, point, etc. then it becomes the end point and the previous end point become a Via point for the routing calculation. Note that the points may also be moved up or down in the sequence, before performing the actual calculation, as explained further down.

Clears the current route and all associated list boxes.

Load a set of Routing points, and user set preferences from a file.

Save the current Routing points and associated user preferences to a file.

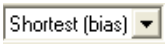
Move the highlighted Routing point to the top of the list, i.e., make it the starting point of the Routing calculation.

Move the highlighted Routing point up one position.

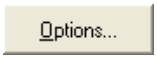
Move the highlighted Routing point down one position.

Move the highlighted Routing point to the bottom of the list, i.e., make it the Finish point of the Routing calculation.

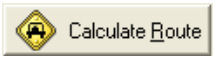
Delete the currently selected Routing point.



Clicking on the down arrow of this combo box, allows the user to select the type of routing calculation to be performed.

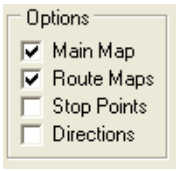


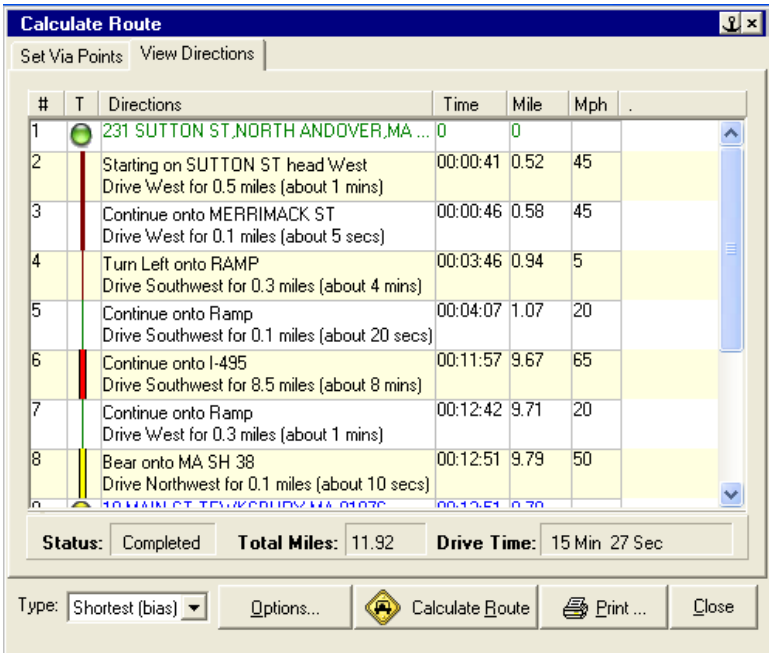
Opens up the Routing options dialog (see below)



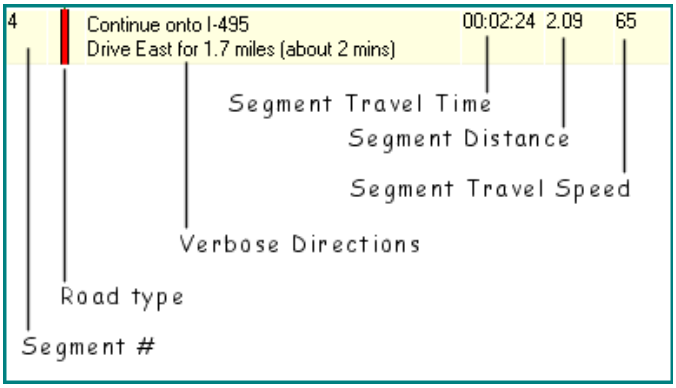
Calculate the route based on the currently selected routing points and options. When the calculation is finished, the dialog automatically switches to the View Directions tab (see below)

Opens up the standard print dialog of the OCX, which allows the user to print the calculated directions. When the print dialog is opened from the router dialog, it contains additional Print options, as shown to the right.



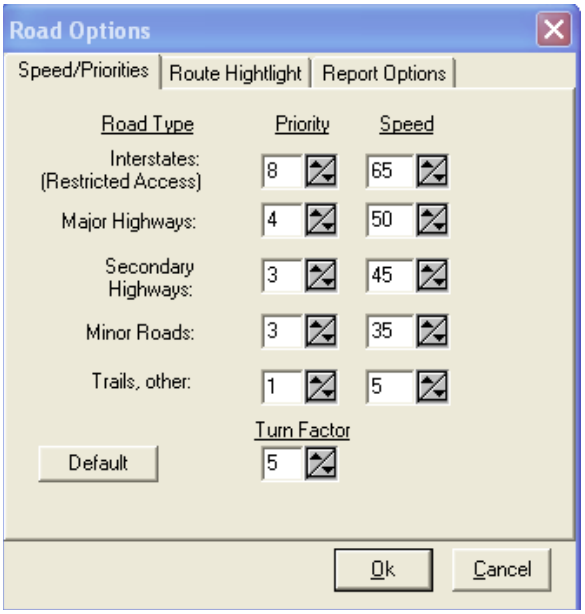


The View Directions tab of the dialog, automatically appears when the routing calculations are finished, and it displays detailed driving directions, as well as the calculated distance and travel times for each segment. The format of each line is as follows…



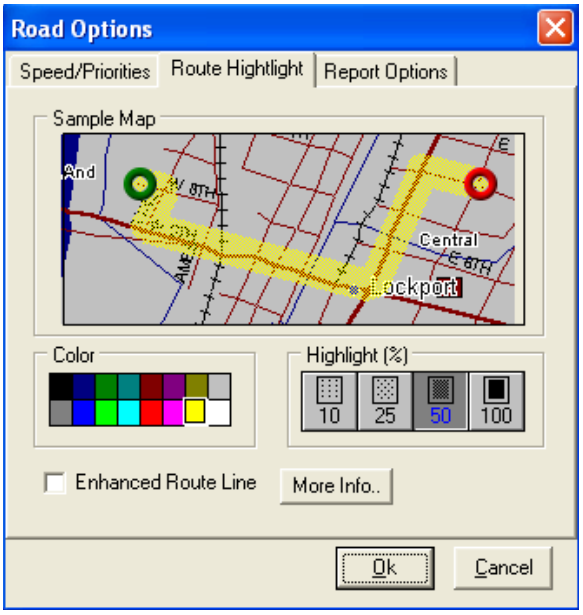
The actual type of information displayed depends on the options selected by the user. The total travel time, distance, etc. are also displayed in a panel, below the segment routing information listbox.

When the user clicks on the options button of the Set Waypoint tab, the following dialog is presented, allowing the user to set a wide range of routing options.



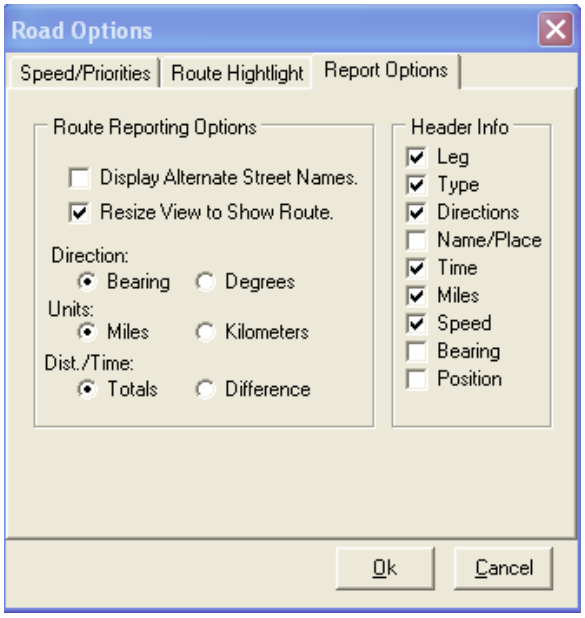
The Speed/Priorities tab allows the user to set the Priority and the travel speed for each major road type. The highest the priority for a certain road type, the more preference is given to that road type during the calculations. Note that even when the priority is set to zero, there is still a non-zero preference used internally, in order to avoid a situation of not being able to calculate a route because of priorities. **Only** priority is used when the user selects the shortest route option.

Turn Factor allows the user to select whether to give preference to a path of fewest turns, over other considered pathways (0-5, 0 uses no biasing). It only has an effect whet the User-Defined routing option is used.



The Route Highlight tab allows the user to select the percent highlight to be used when marking the calculated route on the map surface.

The enhance Route Line option allows the user to bypass some the built-in smoothing of road segments, that is designed to speed up the routing calculations. Note that setting this option ON, can significantly increase the Routing calculation time. Even when it is ON, there is some smoothing done, using the Douglas-Pucker smoothing algorithm. Look at SetOption($EEED,n) for bypassing smoothing completely.



The Report options allows the user to select what information to be displayed in the List box containing the routing information for each Road segment.

The third tab of the Routing Dialog contains brief instruction on how to use it.

VB Example

```
Private Sub Command39_Click()
  MapPro1.ExecRoute
End Sub
```

Delphi Example

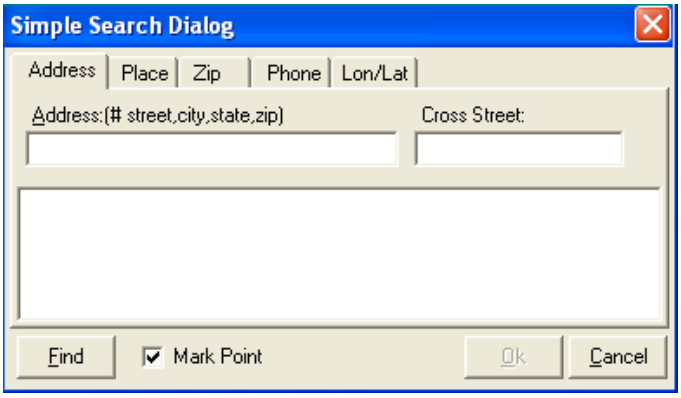
```
Procedure TForm1.Button1Click(Sender: TObject);
{-----------------------------------------------------}
{ Invoke the Routing Dialog with the current default }
{ Lon/Lat Coordinates.                                }
{-----------------------------------------------------}
begin
  { Locate the viewport to the center point of the current}
  { Lon/Lat routing coordinates                          }
  MapPro1.GotoPoint((defx1+defx2)/2,(defy1+defy2)/2);
  { Set the magnitude to 11 to ensure Tiger grids are loaded }
  MapPro1.Magnitude:=11;
  { Execute the Dialog }
  MapPro1.ExecRoute(defx1,defy1,defx2,defy2);
end;
```

ExecSearch(Title:string;Option:Integer) — Procedure

Presents user with a generic tabbed search dialog similar to that used in the Routing dialog.The string specified in "Title" is used in the dialog caption, and 'Option' can have the value of 0..4, depending which tab the user wants active when the dialog opens.

Option=0

Open the dialog with the Address Search tab selected.



Once the dialog appears on the screen, the user may type in an address to search for, in the form:

```
# Street,Place,State,ZipCode
```

And a cross street, if desired. The ZipCode is optional, but it can speed search significantly under certain conditions. In typing the # and Street name, the following forms are equivalent (for example if searching for "10 West Main Street"):

10 W. Main Street
10 West Main Street
10 W Main Street
10 W.Main Street
10 W. Main St
10 West Main St
10 W Main St
10 W.Main St

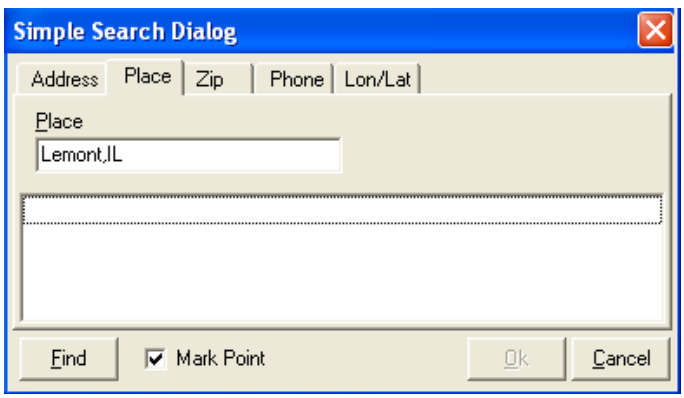
If the suffix is not specified, then all Streets, Avenues, Lanes, etc. meeting the criteria are returned (see Appendix "D" for suffix abbreviations)

Double-clicking on one of the addresses returned in the listbox, will center the viewport around that street segment's Lon/lat and will place the following information in the Result variable (separated by tabs, #9):

```
StreetName,Place,Block#,State,ZipCode,Lon,Lat
```

Option=1

Open the dialog with the Place Search tab selected.



The user may specify a Place, City, Town to search for, including the state (if desired). Wildcard characters are permitted in the search. The only condition is that the first character of either the Place name or the state CANNOT be a wildcard! For example,

(1) Specifying Lem*,IL as the search string would return the following hits:

Lemmon,IL
Lemont,IL

(2) Specifying Lee*,IL as the search string would return the following hits:

Lee,IL
Lee Center,IL
Lee County,IL
Leeds,IL
Leesburg,IL
Leesville,IL

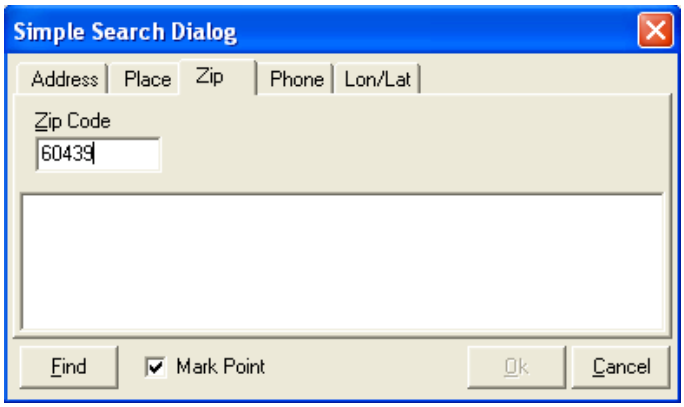
(3) Specifying Le???a,T? as the search string would return the following hits:

Leanna,TN
Levita,TX

Double-clicking on one of the Place names returned in the listbox, will center the viewport around its Lon/lat and will place the following information in the Result variable (separated by tabs, #9):

```
Place,State,Lon,Lat
```

Option=2
Open the dialog with the ZipCode Search tab selected.



The user may enter a ZipCode for the search. If all five digits of the ZipCode are specified, a single hit is returned, if found in the ZipCode database. Wildcards may be used in the search as well.

For example, specifying ?2?14 as the search string would return a number of hits such as,

02814 Chepachet, RI
02914 East Providence, RI
12214 Albany, NY

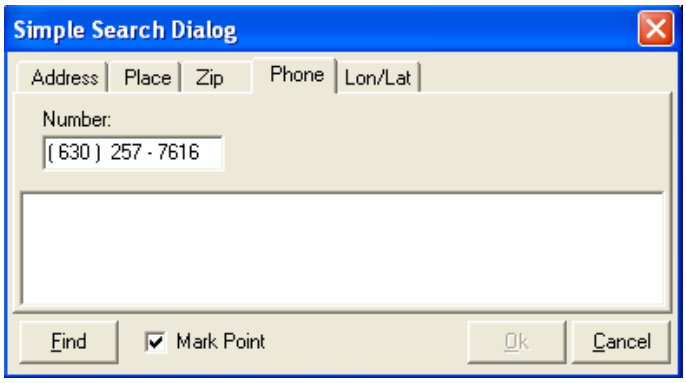
and so on....

Double-clicking on one of the Place names returned in the listbox, will center the viewport around its Lon/lat and will place the following information in the Result variable (separated by tabs, #9):

```
Place,State,AreaCode,ZipCode,Lon,Lat
```

Option=3

Open the dialog with the Area Code Search tab selected.



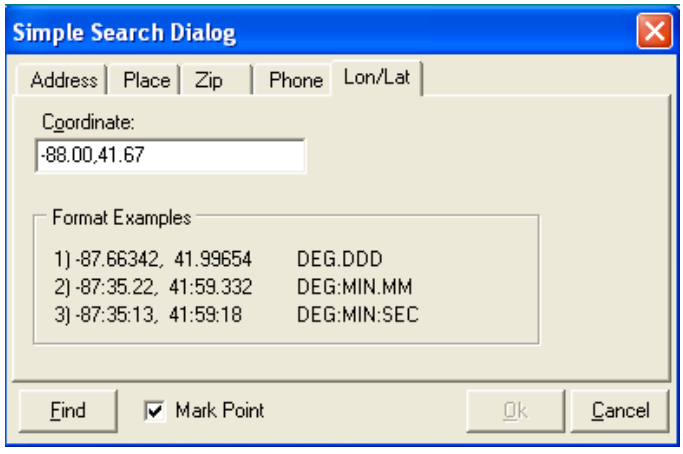
The user may enter an area code and a local exchange for a search. Area code, Exchange, Place Name, State and Zipcode information matching the criteria, are displayed in the listbox.

Double-clicking on one of the entries returned in the listbox, will center the viewport around its Lon/lat and will place the following information in the Result variable (separated by tabs, #9):

```
Place,State,ZipCode,AreaCode&Exchange,Lon,Lat
```

Option=4

Open the dialog with the Lon/Lat Search tab selected.



The user may enter a Lon/Lat coordinate separated by comma, (see the Coords property for changing the input to Lat/Lon). Pressing ENTER or clicking Find centers the viewport around the specified point.

VB Example

```
Private Sub Command39_Click()
  MapPro1.ExecSearch "My Search Dialog",4
End Sub
```

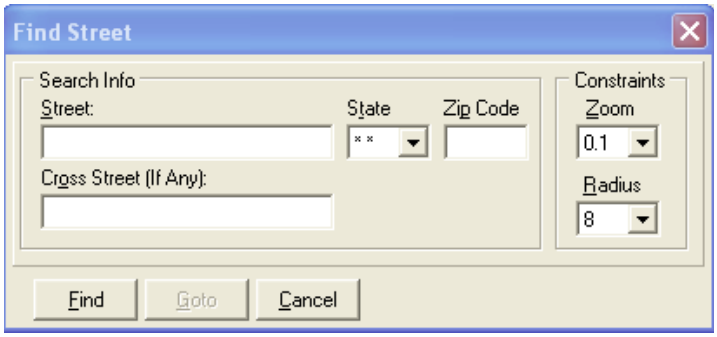
Delphi Example

```
Procedure TForm1.Button4Click(Sender: TObject);
//-----------------------------------------------
// Search for a street
//-----------------------------------------------
begin
  MapPro1.execsearch('Searching for a Street',0);
end;

Procedure TForm1.Button1Click(Sender: TObject);
//-----------------------------------------------
// Search for a Place
//-----------------------------------------------
begin
  MapPro1.execsearch('Searching for a Place',1);
end;
```

ExecStreet() — Procedure

Presents user with a street search dialog (similar to that in Precision Mapping 4.0). Once a street is specified and the search is completed, the user is presented with a listbox containing the retrieved matches. Selecting one of the listbox choices presented, by double clicking, will center the viewport at the Lon/Lat of the selected street segment.

Note that a cross Street may also be specified in the search. Also, note that the user can specify the search radius (Search is faster for smaller search radii), and the State (default is **, which signifies search ALL states within the specified radius).



VB Example

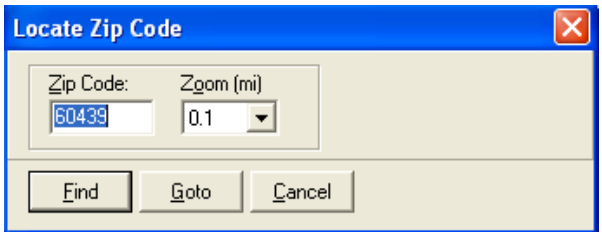
```
Private Sub Command39_Click()
  MapPro1.ExecStreet
End Sub
```

Delphi Example

```
Procedure TForm1.Street1Click(Sender: TObject);
begin
  MapPro1.ExecStreet;
end;
```

ExecZipcode() Procedure

Presents user with the standard Precision Mapping Streets zipcode search dialog. Once a ZipCode is specified and the search is completed, the user is presented with a listbox containing the retrieved matches. Selecting one of the listbox choices presented, by double clicking, will center the viewport at the Lon/Lat of the selected ZipCode centroid.



VB Example

```
Private Sub Command39_Click()
  MapPro1.ExecZipcode
End Sub
```

Delphi Example

```
Procedure TForm1.Zipcode1Click(Sender: TObject);
begin
  MapPro1.ExecZipcode;
end;
```

FindAcEx(temp:String) Procedure

Searches the Area Code/Exchange data base (FONE.BIN) and returns the area code/exchange and other data closest to the query value in the 'Result' string. 'Result' is a variable length string formatted as follows;

```
Areacode #9 Exchange #9 City #9 State #9 Lon #9 Lat
```

The argument Temp can be 3 or 6 digits in length with the first 3 digits taken as the area code. If 4 or 5 digits are specified, then the only 3 are used and the first exchange in the specified area code is returned.

Notes: The location of the phone data is specified in the property 'path3'. It is up to the user to check the first 6 characters returned by the OCX to determine if the desired area code and/or exchange were found.

It should also be noted that no "OnFind" event is triggered by this procedure. The "Result" property should be checked immediately after the call to it. If 'AreaCode' in Result is the same as that specified by the user for the search, then the requested AreaCode was found, if the Areacode returned is different that that specified, then no exact match occurred, and the closed match (next higher AreaCode) was returned.

VB Example

```
Private Sub Command39_Click()
  MapPro1.FindAcEx Text1.text
  Text2.text = MapPro1.Result
End Sub
```

Delphi Example

```
Procedure TForm2.Button6Click(Sender: TObject);
begin
  MapPro1.findacex(edit5.text);
  label7.caption:=MapPro1.Result;
end;
```

FindCity(temp:String) — Procedure

Searches the Place data base (CITIES03.BIN) and returns the place specified in temp, by generating an OnFindPlace event. The search focus is better if the desired place name is followed by a comma and the corresponding state abbreviation. This method will also accept wildcards both in the City/Place name and the two letter State abbreviation. The only condition is that the first character of neither the Place name or the state abbreviation CANNOT be a wildcard.

For each place that is found matching the search specification an OnFindPlace event is triggered. The user must then query the 'Result' property in the OnFindPlace event handler, and store it or otherwise process its value, e.g., add it to a list box, parse it to its appropriate components. The string stored in "Result" contains the following information separated by a tab character (#9).

In addition to being able to search for Place names, this method also enables the user to search for counties (or parishes, in LA). Counties may be specifically searched for by appending the suffix "county", after them, or they cab be search as part of a wildcard place search, by simply specifying the name and an asterisk.

FindCity supports two different paradigms:

- If an explicit string is searched for, then the OnFindPlace is triggered and a single hit (if found) returns in the "Result" property of the control.
- If a wildcard search is specified, then an OnFind event is triggerred and found hits are returned in the "Street" property.

Name: The place (City, Town, etc.) name from the places database
State: The two letter state abbreviation
Lon: Longitude (from the places database)
Lat: Latitude (from the places database)

Examples of using the wildcards with FindCity

(1) FindCity('Lem*,IL') would return the following hits in the Result property

- Lemmon IL -89.794 39.531
- Lemont IL -88.002 41.674

(2) FindCity('Lee*,IL') would return the following hits in the Result property

- Lee IL -88.941 41.795
- Lee Center IL -89.279 41.747
- Lee County IL -89.283 41.751
- Leeds IL -88.988 41.021
- Leesburg IL -90.317 40.247
- Leesville IL -87.625 41.025

(3) FindCity('Lem*,I?') would return the following hits in the Result property

- Lemhi ID -113.619 44.852
- Lemhi County ID -114.017 44.967
- Lemhi Range ID -113.490 44.519
- Lemmon IL -89.794 39.531
- Lemont IL -88.002 41.674

(4) FindCity('Lemont*') would return the following hits in the Result property

- Lemont IL -88.002 41.674
- Lemont PA -77.819 40.810
- Lemont Furnace PA -79.670 39.914
- Lemontree AZ -111.743 33.410
- Lemontree Condomiium UT -111.876 40.901

(5) FindCity('Le?n??t') would return the following hits in the Result property

- Leinarts TN -84.190 36.108
- Leoncito NM -105.138 34.672

(6) FindCity('Le????t,T?') would return the following hits in the Result property

- Le Verte TX -93.927 30.529
- Ledbetter TN -88.729 35.934
- Ledbetter TX -96.791 30.151
- Ledbetter Hills TX -96.926 32.692
- Lee Estates TN -84.886 35.223
- Lees Station TN -85.253 35.554
- Leggett TX -94.870 30.818

VB Example

```
Private Sub Command39_Click()
  MapPro1.FindCity Text1.text
  Text2.text = MapPro1.Result
End Sub
```

Example of how this works:

```
** Explicit Search **
     FindCity Search String: Lemont
            Event Triggerred: OnFindPlace
    Number of hits Returned: 1
   Property Used for Hit(s): Result
            Hit(s) Returned: LEMONT | IL | -88.002,41.674

** WildCat Search **
    FindCity Search String: Lemont,*
           Event Triggerred: OnFind
   Number of hits Returned: 3
  Property Used for Hit(s): Result & Street
            Hit(s) Returned:  LEMONT | IL | -88.002,41.674
                              LEMONT | IA | -94.469,41.333
                              LEMONT | PA | -77.819,40.811
```

VB Example

```
Private Sub Command113_Click()
  List1.Clear
  ' This wll return all Essex, e.g. Essex County, Essex Falls,
  ‘ etc. in MA
  List1.AddItem "Wildcard - Essex, in MA"
  MapPro1.FindCity "Essex*,MA"
  ' This will return ONLY "Essex County" in ALL states
  List1.AddItem "Essex County in ALL States"
  MapPro1.FindCity "Essex County,*"
End Sub
```

Delphi Example

```
Procedure TForm2.Button5Click(Sender: TObject);
begin
  MapPro1.findCity(edit4.text);
  label7.caption:=MapPro1.Result;
end;
```

FindClosest(X,Y,Radius:OleVariant) Procedure

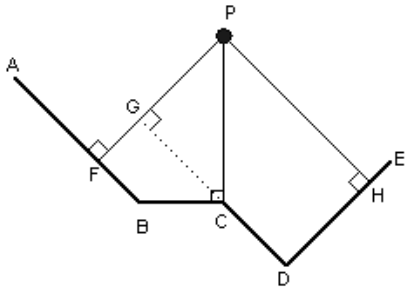
Searches the Streets data bases within the specified radius, and finds the street segment that is closest to the user specified point X,Y (Lon/Lat). The radius (miles) is useful in restricting the search operation which, if not bound, could take a considerable length of time. The procedure performs an exhaustive search, and calculates the distances to every street in the grids within "Radius", even if only portion of such a grid is within that radius.

The results of the search are returned in the property "Result", which a string that contains the following fields separated by character #9:

```
Street, City, Block#, State, Zip, Lon, Lat, Distance
```

The "Lon, Lat" in Result is the point on the street segment determined to be the closest to the user specified point. If the Search fails, a null string is returned as the Street name in Result. *See SetOption with OpCode $EEEE for selecting either the closest road segment end-point, or the closest interpolated point.*

Note: It should be noted that the perpendicular distance from the specified point to each road segment is used in determining the closest one.



For example, if AD, BC, CD, DE are the road segments in question, and P is the point of interest, perpendicular distances PG, PF, PC and PH are calculated and compared to determine the shorter distance, i.e., the closest street segment.

VB Example

```
Private Sub Command39_Click()

  Dim Lon as Double
  Dim Lat as Double
  Dim Temp as String
  Dim Street as String
  Lon = (MapPro1.LonLeft + MapPro1.LonRight) / 2
  Lat = (MapPro1.LatTop = MapPro1.LatBottom) / 2
  MapPro1.FindClosest Lon, Lat, 0.03)
  Temp = MapPro1.Result
  Street = Mid(temp, 1, InStr(1, temp, Chr(9)) - 1)

  For X = 1 To 8
       Temp = Mid(Temp, InStr(1, Temp, Chr(9)) + 1)
       If X = 5 Then Lon = Mid(Temp,1,InStr(1,Temp, Chr(9)) - 1)
       end If
       If X = 6 Then Lat = Mid(Temp,1,InStr(1,Temp, Chr(9)) - 1)
       End If
  Next X
  MapPro1.Miles = 0.2
  MapPro1.drawbubble GetDC(MapPro1.handle),Lon,Lat,Street
End Sub
```

Delphi Example

```
Procedure TForm1.Button49Click(Sender: TObject);
{-------------------------------------------------}
{ Find the closest street to the current viewport  }
{ centroid.  Search radius is set to 2 miles}
{-------------------------------------------------}
var j,dc:integer;
    s:string;
    Lon,Lat:real;
begin
  {Calculate current viewport centroid}
  Lon:=(MapPro1.LonLeft+MapPro1.LonRight)/2;
  Lat:=(MapPro1.LatTop+MapPro1.LatBottom)/2;
  { Find the closest street}
  MapPro1.Findclosest(lon,lat,2);
  Application.ProcessMessages;
  {get the lon/lat coordinates from the returned result}


  {and draw a bubble at that location}
  j:=pos(chr(9),MapPro1.result);
  s:=copy(MapPro1.result,1,j-1);
```

```
  MapPro1.miles:=0.2;
  dc:=getdc(MapPro1.handle);
  MapPro1.drawbubble(dc,lon,lat,'Closest is: '+s);
end;
```

FindClosestPlace(X,Y,Rad,Pop,Opt):string — Procedure

Finds the closest named place from a point in the USA. The returned string contains the following information, separated by tab (#9) characters:

```
Place Name,  FIPS Code,  Population,  Distance from specified point,  Longitude & Latitude
```

X,Y: Lon/Lat coordinates - search center point
Rad: Search radius in miles
Pop: Population criterion to be used for search
Opt: Option specifying the type of search,
-1 Find closest place with population less than the one specified
+1 Find closest place with population higher than the one specified
0 Find closest place regardless of population

VB Example

```
Private Sub Command1_Click()
  'Find the closest named place from the center of the screen,
  'with population more than 30000 within 10 miles
  xc = (MapPro1.LonRight + MapPro1.LonLeft) / 2
  yc = (MapPro1.LatTop + MapPro1.LatBottom) / 2
  city = MapPro1.FindClosestPlace(xc, yc, 30000, 10, 1)
  ' Display the returned information in an edit box
  ' Info is: FIPS Code, Pop, Dist, Longitude, Latitude
  ' separated by tab characters
  Text1.Text = city
End Sub
```

Delphi Example

```
procedure TForm1.Button2Click(Sender: TObject);
var xc,yc:real;
begin
  // Find the closest named place from the center of the screen,
  // with population less than that specified in the Edit2 box
  // within the radius specified in the Edit3 box.
  xc:=(pmap21.lonRight+pmap21.lonleft)/2;
  yc:=(pmap21.lattop+pmap21.latbottom)/2;
  City:=pmap21.FindClosestPlace(xc,yc,
        strtoint(edit2.text),strtoint(edit1.text),-1);
  // Display the returned information as a panel caption
  // Info is: FIPS Code, Pop, Dist, Longitude, Latitude
  panel2.caption:=city;
end;
```

FindItem(X,Y:OleVariant) **Function**

Returns the User Item ID#, if one is found within 8 pixels from X,Y (lon, Lat), or zero otherwise. It also triggers and OnFind and returns a Result string with ID,X,Y,ItemString separated by a tab character (#9) (Also see *SetItem* and *FindFirstItem*).

Note: This is a very powerful method that allows you to hot-link desired actions to user placed bitmaps. Although searching for the item ID can be a CPU intensive process, it has been tried with several thousand points on the screen without a significant time penalty.

VB Example

```
Private Sub Command39_Click()
  Dim Lon as Double
  Dim Lat as Double
  Dim ID as Integer
  Dim Temp as String
  Lon = (MapPro1.LonLeft + MapPro1.LonRight) / 2
  Lat = (MapPro1.LatTop = MapPro1.LatBottom) / 2
  MapPro1.Finditem Lon, Lat
  Temp = MapPro1.Result
  ID = Val(Left(Temp, 1))
  MapPro1.DeleteItem (ID)
end Sub
```

Delphi Example

```
Procedure TForm1.MapPro1MouseMove(Sender: TObject;
Shift:TShiftState; X,Y: Integer);
begin
  If MsMode=1 then
  begin
    MapPro1.MapMode:=MdUser;
    MapPro1.Cursor:=crArrow;
  end else
  begin
    if Imode=1 then
    begin
      MapPro1.Mapmode:=MdUser;
    end else
    begin
      Imode:=0;
      fpt:=MapPro1.Finditem(MapPro1.xcord,MapPro1.ycord);
      if fpt=0 then
      begin
        MapPro1.Mapmode:=MdZoom;
      end else
      begin
        Imode:=2;
        MapPro1.mapmode:=MdUser;
        MapPro1.cursor:=crHelp;
      end;
     panel5.caption:='Pt #'+inttostr(fpt);
    end;
  end;
end;
```

FindFirstItem(X,Y:OleVariant):String — Function

Searches for user items within 8 pixels from the specified point, creates a list of the ones it finds, and and returns a string identifying the first in the list. Note thet the bitmap associates with the item and the bounding polygon for the string associated with the item are used in a point-in-polygon calculation to determine what to return as a valid hit. The string contains the following information separated by a tab character.

Item_ID #9 Item_X-Coordinate #9 Item_Y-Coordinate #9 Item_String

This is one of a set of 3 fuctions that work together, also see *FindNextItem* and *FindItemClose*. The developer needs to make sure that once finished they call the FindItemClose method to close the list object and return any used resources to the resource pool.

Delphi Example

```
procedure TForm1.Button11Click(Sender: TObject);
var s:string;
begin
  // Find User Items within 8 pixels from specified point
  s:=Mappro1.FindFirstItem(-76.10011, 42.0001);
  while s<>'' do
  begin
    listbox2.Items.Add(s);
    s:=mappro1.FindNextItem;
  end;
  mappro1.FindItemClose;
end;
```

FindNextItem():String — Function

Once the FindFirstItem function has been called, and the list of found objects is created, this function returns the next item in the list. A null string is returned if no more objects are in the list. The string contains the following information separated by a tab character.

Item_ID #9 Item_X-Coordinate #9 Item_Y-Coordinate #9 Item_String

This is one of a set of 3 fuctions that work together, also see *FindFirstItem* and *FindItemClose*. The developer needs to make sure that once finished they call the FindItemClose method to close the list object and return any used resources to the resource pool.

Delphi Example

```
procedure TForm1.Button11Click(Sender: TObject);
var s:string;
begin
  // Find User Items within 8 pixels from specified point
  s:=Mappro1.FindFirstItem(-76.10011, 42.0001);
  while s<>'' do
  begin
    listbox2.Items.Add(s);
    s:=mappro1.FindNextItem;
  end;
  mappro1.FindItemClose;
end;
```

FindItemclose() **Method**

Once the FindFirstItem and FindNextItem have been used to find all desired user objects at a specified location, this method needs to be called to close the list object and return any used resources to the resource pool.

This is one of a set of 3 fuctions that work together, also see *FindFirstItem* and *FindItemClose*.

FindRoute(Lon1,Lat1,Lon2,Lat2:OleVariant;Opt: LongInt) **Procedure**

Performs a calculation between the points identified by Lon1,Lat1 and Lon2,Lat2. *(Only enabled if RoutingActive is set to True – see info on the RoutingActive property).*

Opt: Long Integer that can accept either an integer value, or one of the enumerated constants described below (or any combination, i.e., sum, of the values of these options)

Enumerated Constant	Value	Action
Rt_Clear	0	Clear the highlighted route
Rt_Spots	1	Mark the Begin and End point of the calculated route.
Rt_Hatch	2	Highlight the calculated route using the current user-selected hatch pattern. (the hatch pattern is selected using the RoadOptions)
Rt_Zoom	4	Zoom the map viewport out so that the route start and end points are visible.
Rt_Short	16	Calculate the Shortest route
Rt_Fast	32	Calculate the Fastest route
Rt_Direct	48	Calculate the most direct route (least # of turns. Note that this is a combination of Rt_Short and Rt_Fast).
Rt_Hours	64	Return the calculated time as a fraction of hr:min, not hr:min:sec.
Rt_km	128	Return the calculated distances in kilometers, not miles.
Rt_Total	512	Display total (cumulative) distance and time at the end of each road segment in the route.
Rt_PrtMap	1024	Print the point maps after the route is calculated
Rt_PrtDir	2048	Print the directions after the route is calculated
Rt_NoERR	4096	Suppresses error dialog from display if no route is found
Rt_NoDlg	8192	Calculate the route without displaying a dialog

For example, if the FindRoute method was called and Options had the value 42, then RT_Hatch, Rt_Combine and Rt_Total would be the selected options. The 'Result' string returned on each OnFindRte event contains the following fields of information separated by a tab (#9) character.

Route Segment No.	- First one is #1
Road Classification	- A41, etc.
Reserved	- Contains % sign, used for other intermediate calculations
Road Name	- First one is always "Start"
Reserved	- Blank
Place name	- City, Town, etc. It should be noted that limited access Interstate highways have no place names assigned to them.
Time	- xx:xx:xx
Reserved	- Blank
Reserved	- Blank
Speed	- Speed for the road segment in mph or kph.
Direction	- E, NW, W, etc.
Position	- Lon/Lat coordinates of segment endpoint.

It should be noted the first string returned when the event is fired contains the literal 'Start' in the third field, while the last string returned, at the completion of the route calculation, contains the literal 'Finish' in the third field.

Note: For each road segment that is found to be part of the calculated route, an OnFindRoute Event is triggered. The user must then query the 'Result' property in the OnFindRoute event handler, and store it or otherwise process this value, e.g., add it to a list box. The user may also parse the returned string into its appropriate components.

Since segments of the same road are consolidated for generating the driving directions, an OnFindDir event is fired every time a consolidated segment is generated, and a tabbed string is passed with it containing the following information:

```
RoadName,Lon,Lat,Descriptor,Direction,Distance
```

VB Example

```
Private Sub Command39_Click()
  Dim FRteOption as Integer
  Dim StartLon as Long
  Dim StartLat as Long
  Dim EndLon as Long
  Dim EndLat as Long
  MapPro1.RoadOption(48,5)
  MapPro1.RoadOption(49,0)

  'Set the desired value for the Options Flag
   FRteOption:=FRteOption+1    {mark the start/finish points}
   FRteOption:=FRteOption+2    {highlight the calculated route}
   FRteOption:=FRteOption+3    {Zoom so endpoints are visible}
   FRteOption:=FRteOption+32   {Return cumulative distances/times)
   FRteOption:=FRteOption+128  {return distances in km)
   'Calculate the Route
   MapPro1.FindRoute StartLon,StartLat,EndLon,EndLat,FRteOption
```

```
   'Refresh the map control
   MapPro1.Refresh
End Sub
```

Delphi Example (Calculating the Route)

```
Procedure TForm1.Button2Click(Sender: TObject);
{-------------------------------------------------------}
{ Set color and hatch options and calculate simple Route }
{ using the FIndRoute method (results handled by user)   }
{-------------------------------------------------------}
const FRteOption:longint=0;
begin
  {Clear the list box, prepare for the new road segments}
  listbox1.Clear;
  {Set color and the hatch pattern for highlighting Route}
  MapPro1.RoadOption(48,5);
  MapPro1.RoadOption(49,0);
  {Set the desired value for the Options Flag}

  FRteOption:=FRteOption+1;    {mark the start/finish}
  FRteOption:=FRteOption+2;    {highlight the route}
  FRteOption:=FRteOption+3;    {Zoom to see route}
  FRteOption:=FRteOption+32;   {Cumulative distances/times)
  FRteOption:=FRteOption+128;  {return distances in km)
  { Calculate the Route }
  MapPro1.FindRoute(defx1,defy1,defx2,defy2,FRteOption);
  { Refresh the map control }
  MapPro1.Refresh;
end;
```

68

```
Example  (Calculating the Route & Printing)
Procedure TForm1.Button2Click(Sender: TObject);
{-------------------------------------------------------}
{ Set options, Calculate and Print Route                 }
{-------------------------------------------------------}
const FRteOption:longint=0;
begin
  listbox1.Clear;
  MapPro1.RoadOption(48,5);
  MapPro1.RoadOption(49,0);
  FRteOption:=FRteOption+1+2+3;
  {Set flag to Print Calculated Route}
  FRteOption:=FRteOption+256;
  MapPro1.FindRoute(defx1,defy1,defx2,defy2,FRteOption);
  MapPro1.Refresh;
end;

Example (placing the results in a listbox)
Procedure TForm1.MapPro1FindRte(Sender: TObject);
{-----------------------------------------------------}
{ Add the road segments to  a simple list box         }
{-----------------------------------------------------}
begin
  Listbox1.Items.add(MapPro1.result);
end;
```

FindViaRoute(Opt: LongInt) **Procedure**

Performs a Routing calculation between the Via points currently defined (note, this is different than FIndRoute which calculates a route between the points identified by Lon1,Lat1 and Lon2,Lat2, as part of the call to the method). See AddViaPoints and related routines on how to add such points to the routing list prior to calling this function.

Opt: Long Integer that can accept either an integer value, or one of the enumerated constants described below (or any combination, i.e., sum, of the values of these options)

Enumerated Constant	Value	Action
Rt_Clear	0	Clear the highlighted route
Rt_Spots	1	Mark the Begin and End point of the calculated route.
Rt_Hatch	2	Highlight the calculated route using the current user-selected hatch pattern. (the hatch pattern is selected using the RoadOptions)
Rt_Zoom	4	Zoom the map viewport out so that the route start and end points are visible.
Rt_Short	16	Calculate the Shortest route
Rt_Fast	32	Calculate the Fastest route
Rt_Direct	48	Calculate the most direct route (least # of turns. Note that this is a combination of Rt_Short and Rt_Fast).
Rt_Hours	64	Return the calculated time as a fraction of hr:min, not hr:min:sec.
Rt_km	128	Return the calculated distances in kilometers, not miles.
Rt_Total	512	Display total (cumulative) distance and time at the end of each road segment in the route.
Rt_PrtMap	1024	Print the point maps after the route is calculated
Rt_PrtDir	2048	Print the directions after the route is calculated
Rt_NoERR	4096	Suppresses error dialog from display if no route is found
Rt_NoDlg	8192	Calculate the route without displaying a dialog

For example, if the FindViaRoute method was called and Options had the value 42, then RT_Hatch, Rt_Combine and Rt_Total would be the selected options. The 'Result' string returned on each **OnFindRte** event contains the following fields of information separated by a tab (#9) character.

Route Segment No.	- First one is #1
Road Classification	- A41, etc.
Reserved	- Contains % sign, used for other intermediate calculations
Road Name	- First one is always "Start"
Reserved	- Blank

Place name	- City, Town, etc. It should be noted that limited access Interstate highways have no place names assigned to them.
Time	- xx:xx:xx
Reserved	- Blank
Reserved	- Blank
Speed	- Speed for the road segment in mph or kph.
Direction	- E, NW, W, etc.
Position	- Lon/Lat coordinates of segment endpoint.

It should be noted the first string returned when the event is fired contains the literal 'Start' in the third field, while the last string returned, at the completion of the route calculation, contains the literal 'Finish' in the third field.

Note: For each road segment that is found to be part of the calculated route, an **OnFindRte** Event is triggered. The user must then query the 'Result' property in the OnFindRte event handler, and store it or otherwise process this value, e.g., add it to a list box. The user may also parse the returned string into its appropriate components.

Since segments of the same road are consolidated for generating the driving directions, an OnFindDir event is fired every time a consolidated segment is generated, and a tabbed string is passed with it containing the following information:

```
RoadName,Lon,Lat,Descriptor,Direction,Distance
```

VB Example

```
Private Sub Command85_Click()
  ‘ Add some Via points
  MapPro1.AddViaPoint -89.4, 43, "MyVia"
  MapPro1.AddViaPoint -112.4, 42.12, "MyVia2"
  MapPro1.AddViaPoint -118.23, 35.55, "MyVia3"
  ' Calculate Route using defined points
  MapPro1.FindViaRoute 31
End Sub
```

FindStr(const t, t2: string;add: Integer;xctr, yctr, mradius: OleVariant;const ststr: string)

Initiates a substring search for the specified street and optional cross street, optional address, center of search (lon/lat) and search radius (miles) and optional state specifier. (also see Street and onFind).

T	- specified street
T2	- optional cross street
Add	- optional address
xctr,yctr	- center of search (lon/lat)
Mradius	- search radius (miles)
Ststr	- optional state specifier (if blank searches all states, otherwise uses two character state abbreviation, i.e. MA;

Note: For each street that is found an OnFind Event is triggered. The user must then query the 'Street' property in the OnFind event handler, and store it or otherwise process this value, e.g., add it to a list box. The user is also responsible for parsing the returned string into its appropriate components.

If a cross street search is requested, the user must handle every OnFind event (one is triggered for every street hit - not just the cross street), and determine whether a cross street was found. This determination can be made by examining the first character of the returned "Street" property. If a cross street was found, that character will be an asterisk. Furthermore, if a cross street was found, the Streetname field will contain a concatenation of the two streets, e.g., "First and Main", and the address block field will contain the string "N/A"

'Street' is a variable length string with each field separated by ASCII code 9 (Tab) and is formatted as follows;

```
StreetName #9 CityName #9 Address #9 State #9 ZipCode #9 Lon #9 Lat #9 Distance.
```

VB Example Example #1 Button routine invokes the search

```
Private Sub Command39_Click()
dim x,y,r as double
  ' Set a sample lat, lon and search radius
  x=-73.5
  y=43.5
  R=20.0

  ' Search for 20 Main street using the sample coordinates
  'specified above }
  MapPro1.Findstr("Main","",20,x,y,r,"MA")

  ' After the search is finished, set the magnification factor
  'to 14 - double lined streets
  MapPro1.Magnitude=14

  ' Since Magnification 14 might involve loading a lot of data,
  'yield to windows message processing, prior to issuing a
  'command to relocate the viewport
  application.ProcessMessages
  MapPro1.GotoPoint(x,y)
End Sub

Example #2   The OnFind Event is used to store and display
          the Street, block number, and City for each hit,
          in a listbox.

procedure TForm1.MapPro1Find(Sender: TObject);
  dim i,l,ns,ne as integer
  dim a as string
  ns=1
  a=""
  l=length(MapPro1.street)
  '{ Use simple loop to extract desired info }
  For i=1 to 3
  ne=pos(#9,copy(MapPro1.street,ns,l-ns-1))
  a=a+copy(MapPro1.street,ns,ne-1)+"  *  "
  ns=ns+ne
  a=a+copy(MapPro1.street,ns,l-ns-1)
  ListBox1.AddItem(a)
End Sub
```

Delphi Example Example #1 Button routine invokes the search

```
Procedure TForm1.Button14Click(Sender: TObject);
var x,y,r:real;
begin
  { Set a sample lat, lon and search radius }
  x:=-73.5;
  y:=43.5;
  R:=20.0;
  { Search for 20 Main street using the sample coordinates
    specified above }
  MapPro1.Findstr('Main','',20,x,y,r,'MA');
  { After the search is finished, set the magnification factor
    to 10 - double lined streets }
  MapPro1.Magnitude:=14;
  { Since Magnification 14 might involve loading a lot of data,
    yield to windows message processing, prior to issuing a
    command to relocate the viewport }
    application.ProcessMessages;
  MapPro1.GotoPoint(x,y);
end;


Example #2  The OnFind Event is used to store and display the
          sreet, block number, and City for each hit, in a
          listbox.

Procedure TForm1.MapPro1Find(Sender: TObject);
var i,l,ns,ne:integer;
    a:string;
begin
  ns:=1;
  a:='';
  l:=length(MapPro1.street);
  { Use simple loop to extract desired info }
  For i:=1 to 3 do
  begin
    ne:=pos(#9,copy(MapPro1.street,ns,l-ns-1));
    a:=a+copy(MapPro1.street,ns,ne-1)+'  *  ';
    ns:=ns+ne;
  end;
  a:=a+copy(MapPro1.street,ns,l-ns-1);
  ListBox1.Items.Add(a);
end;
```

FindZip(z:LongInt) — Procedure

Searches the zipcode database (ZIP.BIN) and returns data associated with the closest zipcode which matches the search query
in the 'Result' property string.

'Result' is a variable length string with each field separated by ASCII code 9 and is formatted as follows:

Zipcode #9 city #9 State #9 areacode #9 lon #9 lat.

The specified Zip Code should be 5 digits long. If less than 5 digits are specified, then the entered value is padded on the left with zeroes, e.g., if 1234 is specified, then the routine searches for the Zip Code 01234.

Note: The location of Zipcode data is specified in the OCX property 'path3'. The areacode in the returned Result is the one associated with the centroid of the ZipCode.

Note that an "OnFind" event is not triggered by this procedure. The "Result" property should be checked immediately after the call to it. If 'ZipCode' in Result is the same as that specified by the user for the search, then the requested Zipcode was found, if the Zipcode returned is different that that specified, then no exact match occurred, and the closed match (next higher ZipCode) was returned.

VB Example

```
Private Sub Command39_Click()
  MapPro1.findzip val(Text1.text)
  list1.additem MapPro1.Result
End Sub
```

Delphi Example

```
Procedure TForm2.Button4Click(Sender: TObject);
var z,code:longint;
begin
  val(edit3.text,z,code);
  MapPro1.findzip(z);
  label7.caption:=MapPro1.Result;
end;
```

FirePmapEvent(EventID, Delay:Integer) **Method**

Fires a generic event added to the control, at the user's will and discretion. The Event is fired by calling the FirePmapEvent method, with the following parameters.

The Event can be fired for multiple operations, using a unique **EventID** specified and managed by the user, and the **Delay**, before the event is fired, is in milliseconds.

When the method is called, the event PmapEvent is fired, with the form:

MapPro1PmapEvent(Sender: TObject; EventID: Integer);

The user needs to manage this event themselves.

Delphi Example

```
Procedure TForm1.Button6Click(Sender: TObject);
// Fire an event with a 500 msec delay and increment the
eventide #
begin
  inc(eid);
  MapPro1.FirePmapEvent(eid,500);
end;

procedure TForm1.MapPro1PmapEvent(Sender: TObject; EventID:
Integer);
// When the event is fired, it is trapped here an a string is
added
```

```
// to the listbox to let the user know
begin
  ListBox1.items.add('Fired Event #'+inttostr(Eventid));
end;
```

Font:String — Property

Inherited Property. Sets the font (Font.Name) and height (font.Height) to be used for State names, Country names, Landmarks, Major water bodies, City names and Highway shields.

Note: If the specified font name is not found on the system, the last font used is assumed.

VB Example

```
Private Sub Command39_Click()
  ' button to set a new font
  Font.Name="Arial"
  Font.Height=16
End Sub
```

Delphi Example

```
Procedure TForm1.Button3Click(Sender: TObject);
begin
  { button to set a new font }
  Font.Name:='Arial';
  Font.Height:=16;
end;
```

GeoFind(s:String):Integer; — Function

Returns the number of hits (if a StreetAddress Search is done), or -1 if a City Search is performed and there is NO unique match to the specified city/place. Each hit generated by the function trigers a Find (OnFind) event and the developer can examine the results for each hit by examining the Street property of the OCX, in the OnFind event. The following fields are returned in the Street property, separated by tab characters #9:

```
Street|City|Block Number|State|ZipCode|*reserved|Longitude,Latitude
```

The *Reserved field is blank for now.

It requires a variable string argument as follows, s = StreetAddress, City, State, ZipCode [|Radius]

Any of the four fields may be ommitted. In particular, the last argument, separated from the rest of the string by a piping character, specifies the search radius (in miles), from the center search point (Zip, state, place coordinates). The search hierarchy is

(1) ZipCode
(2) City, State
(3) StreetAddress

Here are some examples and brief explanations of the operation of the function. A few of them are explained in detail, while the rest are presented as the specified string and a simple listing of the results returned in the Street property.

N = GeoFind('Lowell,MA')

Returns the N = 1 (one hit) and examination of the Street property reveals the string:

|LOWELL||MA|||-71.317000,42.633000

where | is used to denote the tab delimeter #9 character, used in the returned string.

The program recognizes that a City, State search is being requested and performs it accordingly.

N = GeoFind('Market St,Lowell,Ma')

Returns the N = 16 (16 hits) and examination of the Street property, on each OnFind event, reveals the strings:

|MARKET ST|LOWELL|630 - 699|MA|01854||-71.318912,42.647360
|MARKET ST|LOWELL|612 - 629|MA|01854||-71.318464,42.647168
|MARKET ST|LOWELL|578 - 611|MA|01854||-71.318144,42.647104
|MARKET ST|LOWELL|564 - 577|MA|01854||-71.317888,42.646912
|MARKET ST|LOWELL|544 - 563|MA|01854||-71.317696,42.646848
|MARKET ST|LOWELL|530 - 543|MA|01854||-71.317504,42.646720
|MARKET ST|LOWELL|512 - 529|MA|01854||-71.317312,42.646656
|MARKET ST|LOWELL|452 - 511|MA|01854||-71.316800,42.646464
|MARKET ST|LOWELL|380 - 451|MA|01854||-71.315776,42.645888
|MARKET ST|LOWELL|353 - 379|MA|01854||-71.315072,42.645632
|MARKET ST|LOWELL|320 - 352|MA|01852||-71.314752,42.645440
|MARKET ST|LOWELL|292 - 323|MA|01852||-71.314048,42.645120
|MARKET ST|LOWELL|248 - 290|MA|01852||-71.313536,42.644928
|MARKET ST|LOWELL|221 - 246|MA|01852||-71.313088,42.644800
|MARKET ST|LOWELL|116 - 219|MA|01852||-71.311872,42.644672
|MARKET ST|LOWELL|1 - 98|MA|01852||-71.309632,42.644544

The program recognizes that a street search (without a block number) is being requested and returns all street segments (blocks) that match the specified street within the city/state specified by the user. Identical results would be returned if the user specified 'Market,Lowell,MA' as well.

However, if the specified search was:

N = GeoFind('207 Market Street,Lowell,MA);

Then N=1 (single hit) would be returned and the property Street would countain the string:

MARKET ST|LOWELL|116 - 219|MA|01852||-71.312576,42.644736

N = GeoFind('Andover')

Returns the N = -1 and examination of the Street property, on each OnFind event, reveals the strings:

|ANDOVER||CA|||-120.246080,39.310528
|ANDOVER||CA|||-120.256384,39.301952
|ANDOVER||CT|||-72.370816,41.737216
|ANDOVER||IL|||-90.291968,41.293888
|ANDOVER||IA|||-90.251648,41.979136
|ANDOVER||KS|||-97.136128,37.713920
|ANDOVER||ME|||-70.751680,44.635584
|ANDOVER||MA|||-71.137472,42.658304
|ANDOVER||MN|||-93.291136,45.233344
|ANDOVER||MO|||-93.894976,40.564992
|ANDOVER||NH|||-71.823872,43.436928
|ANDOVER||NJ|||-74.742528,40.985856
|ANDOVER||NY|||-77.795840,42.156416
|ANDOVER||OH|||-80.572480,41.606656
|ANDOVER||PA|||-78.084160,39.946368
|ANDOVER||SC|||-82.205824,35.079168
|ANDOVER||SD|||-97.902208,45.410304
|ANDOVER||VT|||-72.697216,43.277248
|ANDOVER||VA|||-82.796672,36.923584
|ANDOVER ESTATES||MD|||-76.504704,38.168640
|ANDOVER GOLF ESTATES||FL|||-80.208640,25.963904
|ANDOVER JUNCTION||NJ|||-74.746112,40.996928
|ANDOVER LAKESOUTH ESTATES||FL|||-80.202240,25.965824
|ANDOVER NORTH||GA|||-84.302528,34.09446

The program recognizes that a search for a place (City, town, etc.) is being performed, but since there is no state or Zip code information, it does a sub-string search and returns all places found that contain the string 'Andover' in them.

N = GeoFind('Andover,01810');

Returns N = 1

|ANDOVER||MA|01810||-71.155800,42.648700

The ZipCode is used for the search.

N = GeoFind('Andover,MA,02117');

Returns N = 1

|BOSTON||MA|02117||-71.060300,42.358300

The ZipCode is used for the search. The correct coordinates (Boston) are returned, and the city/state specification is ignored.

N = GeoFind('Main Street,Andover,MA');

Returns N = 11

MAIN ST|ANDOVER|1 - 4|MA|01810||-71.140096,42.656896

MAIN ST|ANDOVER|5 - 17|MA|01810||-71.139904,42.656512
MAIN ST|ANDOVER|33 - 46|MA|01810||-71.139712,42.656128
MAIN ST|ANDOVER|47 - 64|MA|01810||-71.139456,42.655680
MAIN ST|ANDOVER|65 - 96|MA|01810||-71.138752,42.654720
MAIN ST|ANDOVER|95 - 108|MA|01810||-71.138048,42.653696
MAIN ST|ANDOVER|109 - 121|MA|01810||-71.137472,42.652864
MAIN ST|ANDOVER|120 - 131|MA|01810||-71.136832,42.651840
MAIN ST|ANDOVER|133 - 158|MA|01810||-71.135936,42.650752
MAIN ST|ANDOVER|151 - 157|MA|01810||-71.135104,42.649600
MAIN ST|ANDOVER|165 - 206|MA|01810||-71.134464,42.648320

All 'Main Street' segments are returned since there is no block number specified. The same results would have been returned if the specified search was:

N = GeoFind('Main St,Andover,01810');

N = GeoFind('1000 Main,Andover,MA');

Returns N = 1

|ANDOVER||MA|||-71.137000,42.658000

N = GeoFind('North Andover');

Returns N = -1

|NORTH ANDOVER||MA|||-71.135552,42.698624
|NORTH ANDOVER||WI|||-90.965824,42.815552
|NORTH ANDOVER CENTER||MA|||-71.112512,42.683328

N = GeoFind('Main Street, North Andover');

Returns N = -1

|NORTH ANDOVER||MA|||-71.135552,42.698624
|NORTH ANDOVER||WI|||-90.965824,42.815552
|NORTH ANDOVER CENTER||MA|||-71.112512,42.683328

Since no unique North Andover was specified, no street search was performed, but a sub-string search on the place was done instead.

N = GeoFind('North Andover,MA,01810');

Returns N = 1
|ANDOVER||MA|01810||-71.155800,42.648700

Results based on ZipCode search.

N = GeoFind('Andover,NH,1045');

Returns N = 1

|ANDOVER||NH|||-71.824000,43.437000

Invalid ZipCode so a City/State search was performed instead.

N = GeoFind('N. andover,MA');

Returns N = 1

|NORTH ANDOVER||MA|||-71.136000,42.699000

N = GeoFind('Lawrence,NH');

Returns N = -1

|LAWRENCE CORNER||NH|||-71.523904,42.847232

A substring search on the city, within the specified state, was performed.

N = GeoFind('Lawrence,MA,01845');

Returns N = 1

|NORTH ANDOVER||MA|01845||-71.117900,42.687600

Results based on ZipCode search

N = GeoFind('Lawrence,MA,02117');

Returns N = 1
|BOSTON||MA|02117||-71.060300,42.358300

GeoFindArray(S:strring); **Method**

It takes the same argument as the GeoFind routine. The results are returned through an interface StreetsArray, through a record structure, IStreetRec. This approach was incorporated to primarily accommodate developers not using visual environments, and therefore not being able to handle events fired by the OCX in order to examine the Street property for hits generated from GeoFind.

```
IStreetRec = record
        Name:string;
        Address:string;
        City:string;
        State:string;
        ZipCode:String;
        AreaCode:String;
        X:double;
        Y:double;
      end;
```

The IStreetsArray Interface object has the following properties & methods.

IStreetsArray.Count	-	# of hits
IStreetsArray.Items[]	-	Indexed array of the unparsed strings that are usually returned by GeoFind
IStreetsArray.Streets[]	-	Indexed list of Parsed records of returned hits (Record Type IStreetRec)

Delphi Example Sample Code (Delphi

```
procedure TForm1.GeoFindHitsClick(Sender: TObject);
//-----------------------------------------------------------------
//   New Method GeoFindArray takes identical arguments to those
//   used by GeoFind and returns an IStreetsArray.
//   IStreetsArray.Count   -  # of hits
//   IStreetsArray.Items[] - Indexed array of the unparsed strings
//                             that are usually returned by GeoFind
//
//   IStreetsArray.Streets[]- ndexed list of Parsed records of
//                           returned hits (Record Type IStreetRec)
//
//     IstreetRec.Name     -  Street Name
//     IstreetRec.Address  -  Block #
//     SIstreetRec.City    -  City Name
//     IstreetRec.State    -  State Name
//     IstreetRec.ZipCode  -  ZipCode
//     IstreetRec.AreaCode -  Telephone Area Code (for Future use)
//     IstreetRec.X        -  Latitude
//     IstreetRec.Y        -  Longitude
//
//     or in terms of the indexed array...
//
//     Streets[n].Name     -  Street Name
//     Streets[n].Address  -  Block #
//     Streets[n].City     -  City Name
//     Streets[n].State    -  State Name
//     Streets[n].ZipCode  -  ZipCode
//     Streets[n].AreaCode -  Telephone Area Code (for Future use)
//     Streets[n].X        -  Latitude
//     Streets[n].Y        -  Longitude
//-----------------------------------------------------------------
--

var  SearchStr:string;
     i:integer;
     gHits:IstreetsArray;
begin
  SearchStr:=edit1.text;
  if length(SearchStr)<3 then
  begin
    SearchStr:='Market Street, Lowell, MA, 01852';
    edit1.text:=SearchStr;
  end;
  // Search for multiple hits by not specifying a block #
  gHits:=MapPro1.GeoFindArray(SearchStr);
  ListBox1.clear;
  listbox1.Items.add('Found '+inttostr(gHits.count)+' Hits');
```

```
  // Note that AC was added to the record for completenes, and
possibly future
  // use.  AC was never returned by the GeoFind method.
  listBox1.items.add('Listed below as: #, Street, City, State,
ZipCode, and AreaCode');
  // List all generated hits
  For i:=0 to gHits.count-1 do
  begin
    Listbox1.items.add(gHits.Streets[i].Address);
    Listbox1.items.add(gHits.Streets[i].Name);
    Listbox1.items.add(gHits.streets[i].City);
    Listbox1.items.add(gHits.streets[i].State);
    Listbox1.items.add(gHits.streets[i].ZipCode);
    Listbox1.items.add(gHits.streets[i].AreaCode+'(N/A)');
    ListBox1.items.add(floattostr(gHits.streets[i].x)
                +','+floattostr(gHits.streets[i].y));
    ListBox1.items.add('---------------------------');
  end;
  // List the unparsed Strings
  ListBox2.clear;
  Listbox2.items.add('Unparsed strings for each hit');
  for i:=0 to gHits.count-1 do
  begin
    Listbox2.Items.Add(gHits.items[i]);
  end;
  Messagebeep(0);
end;
```

GeoFindClose() — Procedure

Clears the list object and returns all used memory to the pool. This should be called after GeoFindFirst/Next metods are used. (See GeoFindFirst for sample source code).

GeoFindFirst(GeofindString:String):String — Procedure

Searches for the specified criteria and creates a list object to hold the results and returns the first hit in the list. Returns an empty string if no further items exist in the list object. The returned result is a tab-delimited string with the following information:

Street #9 City #9 Block #9 State #9 ZipCode #9 Reserved #9 Lon,Lat

VB Example

```
Private Sub Command1_Click()
  If FirstTime = True Then
      s = MapPro1.GeoFindFirst("Winter Street, Boston, MA")
      FirstTime = False
      nfd = 1
  Else
      nfd = nfd + 1
      s = MapPro1.GeoFindNext
  End If
  If s <> Null Then
      Label4.Caption = "[" & Str(nfd) & "] " & s
  Else
```

```
        Label4.Caption = "No more Items Found"
        MapPro1.GeoFindClose
    End If
Label4.Visible = True
    Beep
End Sub
```

GeoFindNext():String — Procedure

Finds and returns the next item in the Geofind list object. Returns an empty string if no further items exist in the list.

VB Example

```
Private Sub Command1_Click()
    If FirstTime = True Then
        s = MapPro1.GeoFindFirst("Winter Street, Boston, MA")
        FirstTime = False
        nfd = 1
    Else
        nfd = nfd + 1
        s = MapPro1.GeoFindNext
    End If
    If s <> nil Then
        Label20.Caption = "[" & Str(nfd) & "] " & s
    Else
        Label20.Caption = "No more Items Found"
```

GeoFindParse(Field, Result:String):String — Procedure

Return the value of Field, from the result string generated by the GeoFindFirst or GeoFindNext methods.

Field The field whose value is to be returned. It can have one of these string values, STREET, CITY, ADDRESS, STATE, ZIPCODE, AREACODE, X, Y

Result The GeofFind result string being parsed.

VB Example

```
Private Sub Command21_Click()
    If FirstTime = True Then
       s = MapPro1.GeoFindFirst(Text8.Text)
       FirstTime = False
       nfd = 1
    Else
       nfd = nfd + 1
       s = MapPro1.GeoFindNext
    End If
    If s <> Null Then
       Label20.Caption = "[" & Str(nfd) & "] " & s
    Else
       Label20.Caption = "No more Items Found"
    End If
    Label6.Caption = MapPro1.GeoFindParse("ADDRESS", s)
```

```
  Label8.Caption = MapPro1.GeoFindParse("STREET", s)
  Label10.Caption = MapPro1.GeoFindParse("CITY", s)
  Label12.Caption = MapPro1.GeoFindParse("STATE", s)
  Label14.Caption = MapPro1.GeoFindParse("ZIPCODE", s)
  Label16.Caption = MapPro1.GeoFindParse("AREACODE", s)
  Label18.Caption = MapPro1.GeoFindParse("X", s) & "," &
  MapPro1.GeoFindParse("Y", s)
  Beep
End Sub
```

GetProductCode:String — Property

Returns the string that is needed to transmit to Undertow in order to register a product. Normally this is not required as the OCX checks to determine if it is registered, and if not it initiates its own internal registration process and presents this information to the end user through its own dialogs. (Also see ExecRegister)

VB Example

```
Private Sub Command21_Click()
  S = Mappro1.GetProductCode
End Sub
```

GetRouteFirst(x1,y1,x2,y2:double; Consolidate:boolean; Option:integer):String

A List object added to the control to facilitate the calculation of routes in non-visual environments, that do not support event-driven operations. (Three methods were added to access this list object, GetRouteFirst, GetRouteNext, and GetRouteClose)

Where: **x1,y1** - are the coordinates of the starting point,
x2,y2 - are the coordinates of the endpoint
Consolidate - a flag that specifies whether to return each individual segment, or consolidate segments with the same name
Option - is an integer that specifies a variety of options to be used for the calculation (see FindRoute for a description of the same variable). It should be pointed out, however, that certain "Option" values that are available as part of the FindRoute method, have no effect when using the "GetRoute..." methods, as hown below:

Enumerated Constant	Value	Action
Rt_Clear	0	Clear Current Route
Rt_Spots	1	Mark the Start and End points
Rt_Hatch	2	Highlight the calculated route
Rt_Zoom	4	No Effect
Rt_Short	16	Calculate the Shortest route
Rt_Fast	32	Calculate the Fastest route
Rt_Direct	48	Calculate the most direct route (least # of turns.

		Note that this is a combination of Rt_Short and Rt_Fast).
Rt_Hours	64	Return the calculated time as a fraction of hr:min, not hr:min:sec.
Rt_km	128	Return the calculated distances in kilometers, not miles.
Rt_Total	512	Display total (cumulative) distance and time at the end of each road segment in the route.
Rt_PrtMap	1024	No Effect
Rt_PrtDir	2048	No Effect
Rt_NoERR	4096	Suppresses error dialog from display if no route is found
Rt_NoDlg	8192	No Effect
Rt_Degree	16384	Return the bearing as degrees, instead of literal.

This function returns the first entry in the routing calculation array, containing information about the first route segment, as a string made up of a number of fields, delimited by the tab character (#).

Here is the information (fields) returned.

Leg	- Integer identifying the # of this segment of the trip.
Type	- The CFCC Code for this road segment
Instructions	- Literal instructions for this Leg
Locale	- The local place & road name
Time	- The time required for this Leg, or the cumulative time, depending on the Option setting.
Dist	- The distance traveled for this Leg, or the cumulative distance, depending on the Option setting.
Speed	- Speed for this Leg (km/hr or mi/hr, based on Option)
Bearing	- Bearing for this Leg (Degrees or literal, based on Option)
Lon or X	- Longitude (or X-coordinate) of Leg start
Lat or Y	- Latitude (or Y-coordinate) of Leg start
Street	- The Street/Road name
Place	- The local place name

It should be noted the first string returned when the event is fired contains the literal 'Start' in the third field, while the last string returned, at the completion of the route calculation, contains the literal 'Finish' in the third field.

GetRouteNext():String — Procedure

Returns the next string of the List Object. If the end of the list has been reached, a blank string is returned.

Delphi Example

```
procedure TForm1.Button5Click(Sender: TObject);
// List Calculated Route segments
var s,t:string;
    i,j:integer;
begin
  i:=0;
  s:=MapPro1.GetRouteFirst(-71.77,42.55, -71.81,42.70, true,
                           Rt_Spots +
                                Rt_Hatch +
                                Rt_Zoom +
```

```
                              Rt_Total+
                              8192);
    messagebeep(0);
    repeat
      listbox1.Items.add('* New Leg');
      inc(i);
      for j:=1 to 12 do
      begin
        t:=MapPro1.GetRouteParse(j,s);
        listbox1.Items.add(t);
      end;
      s:=MapPro1.GetRouteNext;
    until s='';
    MapPro1.GetRouteClose;
  end;
```

GetRouteClose() **Procedure**

Destroys the Routing List Object and returns all resources to the system. Needs to be used once the developer is finished with the GetRouteFirst/Next methods.

RouteParse(Fld:Variant, s:string):String **Procedure**

Parses the results returned by the GetRouteFirst, GetRouteNext, GetOptiFirst, GetOptiNext methods.

Fld - FieldName or Index #
s – StringToBeParsed

Fld is one of the fields (or its equivalent fild #) from the list presented earlier, i.e.,

Here is the information (fields) returned.

No	Name	Explanation
1	Leg	Integer identifying the # of this segment of the trip.
2	Type	The CFCC Code for this road segment
3	Instructions	Literal instructions for this Leg
4	Locale	The local place & road name
5	Time	The time required for this Leg, or the cumulative time, depending on the Option setting.
6	Dist	The distance traveled for this Leg, or the cumulative distance, depending on the Option setting
7	Speed	Speed for this Leg (km/hr or mi/hr, based on Option)
8	Bearing	Bearing for this Leg (Degrees or literal, based on Option)
10	Lon or X	Longitude (or X-coordinate) of Leg start
11	Lat or Y	Latitude (or Y-coordinate) of Leg start
12	Street	The Street/Road name

13	Place	The local place name

For example, GetRouteParse(2,GetRouteNext)) or GetRouteParse(Type,GetRouteNext)) would return the CFCC road type of the current Route Leg.

VB Example

```
Private Function MyQFactor() As Integer
MyQFactor = 1
 If MapPro1.GeoFindParse(3, MapPro1.Street) <> "" Then
   MyQFactor = 5
 Else
   If MapPro1.GeoFindParse(1, MapPro1.Street) <> "" Then
     MyQFactor = 4
   Else
     If MapPro1.GeoFindParse(2, MapPro1.Street) <> "" Then
       MyQFactor = 3
     Else
       If MapPro1.GeoFindParse(5, MapPro1.Street) <> "" Then
         MyQFactor = 2
       Else
         If MapPro1.GeoFindParse(4, MapPro1.Street) <> "" Then
MyQFactor = 1
       End If
     End If
   End If
 End If
End Function
```

GetViaRouteFirst(Consolidate:Boolean; Options:Integer):String Procedure

Used to perform a routing calculation with Via points and return the results in a list object. Consolidate controls whether the similarly named road segments will be consolidated into a continuous segment, and Options is the same Options parameter used in the FindRoute method. This method creates a list object and populates it with the calculated routing instructions, and returns the first one. See GetViaRouteNext and GetViaRouteStop for information on how to get the remaining directions and close the list object.

The returned string is of the same format as the string returned in the GetRouteFirst/Next and can therefore be parsed using the GetRouteParse method.

VB Example

```
Private Sub Command104_Click()
  ' Clear the list object to get the results
  ' of the new routing calculation
  List1.Clear
  ' Get the first routing leg.  This presumes that
  ' Some Via points have been defined.
  If Check1.Value = 1 Then
    s = MapPro1.GetViaRouteFirst(True, 31)
  Else
    s = MapPro1.GetViaRouteFirst(False, 31)
  End If
  ' Continue to get the rest of the routing serments
  While s <> ""
```

```
    List1.AddItem s
    s = MapPro1.GetViaRouteNext
  Wend
  ' Close the list object
  MapPro1.GetViaRouteClose
End Sub
```

GetViaRouteNext():String — Procedure

Returns the next string of the List Object created when GetViaRouteFirst is called. If the end of the list has been reached, a blank string is returned.

GetViaRouteClose():String — Procedure

Closes the list object created when GetViaRouteFirst is called, and releases all resources used by the object back to the system.

GotoPoint(x, y: OleVariant) — Procedure

Centers the viewport around the specified lon/lat coordinate in decimal degrees. The screen is updated after this command. (Note that no change of the scale takes place).

VB Example

```
Private Sub Command1_Click()
   Call MapPro1.GotoPoint(-110,42)
   Or
   MapPro1.GotoPoint -110,42
 End Sub
```

Delphi Example

```
Procedure TForm1.Button7Click(Sender: TObject);
begin
  MapPro1.GotoPoint(-110,42);
end;
```

Grid:Boolean

Makes visible the lon and lat grid lines on the map, with appropriate scale labeling. The labeling depends on the value of the property LLMode (which also controls the display format in the coordinate area of the toolbar). An alternative way to control the baleling of the Grid lines is through the use of the property of degFormat.

Note: It should be noted that when LLMode=3, the grid labels are in Deg.DecimalMinutes, as it would make no sense to label them with screen coordinates.

VB Example

```
Private Sub Command39_Click()
  ' Button to display gridstatus
  MapPro1.Grid=1
```

```
End Sub
```

Delphi Example

```
Procedure TForm1.Button8Click(Sender: TObject);
begin
  // Button to display gridstatus
  MapPro1.Grid:=1;
end;
```

Handle:Integer — Property

Windows Handle of the OCX container. Used by Windows API calls to manage the control.

VB Example

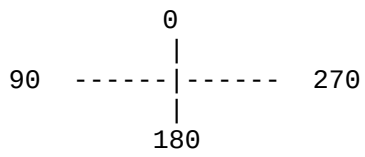
```
Private Sub Command39_Click()
  dc=getdc(MapPro1.handle)
End Sub
```

Delphi Example

```
{-----------------------------------------------------}
{ Draw a line using the control's DrawLine Method      }
{-----------------------------------------------------}
var dc,w,color,mode:longint;
begin
  {get the dc for the map object}
  dc:=getdc(MapPro1.handle);
  {set the color to blue}
  Color:=clblue;
  {set the mode to raster operation to R2_MergePen}
  Mode:=R2_MergePen;
 {set the width to 10 pixels}
  W:=10;
  {Draw a line from Chicago to LA.   Note that if the screen}
  {is updated, the line is not redrawn unless it's tied to}
  {the OnPaintAfter event}
  MapPro1.DrawLine(dc,-87.65,41.84,-118.24,34.05,W,Color,Mode);
  {finally, release the dc}
  releasedc(handle,dc);
end;
```

HeadsUp(x1,y1,x2,y2:OleVariant):longint — Function

Given the coordinates of two points (Lat/Lon), it returns the angle (degrees) that the map must be rotated by, in order for the "PointOne-to-PointTwo" direction to be UP. Note that the zero degree angle is due North, and the positive rotation is counterclockwise. Useful for developers that need to always display direction of movement, for example, always up.



VB Example

```
Private Sub Command39_Click()
  'Use two points on horizontal line, which
  'should rotate the map by 90 degrees
  Dim x as Long
  x = MapPro1.Headsup -101.56,48.12,-101.43,48.12
  MapPro1.Rotate(x)
End Sub
```

Delphi Example

```
Procedure TForm1.Button1(Sender: TObject);
var x:longint;
begin
  { Use two points on horizontal line, which
    should rotate the map by 90 degrees }
  x:=MapPro1.Headsup(-101.56,48.12,-101.43,48.12);
  MapPro1.Rotate(x);
end;
```

HelpPath:String — Property

Specifies the help file/path to be used by Winhelp when the question mark icon on the toolbar is clicked.

VB Example

```
Private Sub Command39_Click()
  MapPro1.helppath="d:\cmap40\help\pmap.hlp"
End Sub
```

Delphi Example

```
Procedure TForm1.FormCreate(Sender: TObject);
begin
  MapPro1.helppath="d:\cmap40\help\pmap.hlp;
end;
```

HideAllItems() — Procedure

Sets the attribute for all user-created item objects to invisible.

VB Example

```
Private Sub Command39_Click()
  MAPPRO1.HideAllItems
End Sub
```

Delphi Example

```
Procedure TForm1.Button9Click(Sender: TObject);
begin
  MapPro1.HideAllItems;
end;
```

Import:ImportLayer; **Property**

Allows the user to access the IImportLayer interface and enables them to load a layer of information from a MID/MIF or a SHP file. Please, see the section **"Import Interface"** of this document for more details.

VB Example

```
Private Sub Command39_Click()
  'Load a shape file as a new layer
   MAPPRO1.import.Filename = "f:\Mydata\Sample.shp"
End Sub
```

ImportMgr:ImportManager; **Property**

Allows the user to access the IImportManager interface and enables them to load and manage a collection of layers of information from numerous MID/MIF or a SHP files. Please, see the section **"ImportManager Interface"** of this document for more details.

VB Example

```
Private Sub Command39_Click()
  'Load a shape file as a new layer
   MAPPRO1.ImportMgr.items(0).filename = "Sample.shp"
End Sub
```

InitNonVis() **Procedure**

It is recommended that this new method is called right after the object is created in a non-visual environment(i.e. ASP). It forces the build-in map (USA with coarse highway network) to be loaded and gets the application handle. Not using this method may result in non-visula applications not displaying any mapping info until after the first need to update the screen, due to a zoom operation, etc.

Int2Lat(i:Integer):Double **Function**

Returns a real value of the Latitude, given the Y coordinate in screen units.

Note: It only applies to non-rotated maps. An incorrect coordinate will be returned if the map has been rotated.

VB Example

```
Private Sub Command39_Click()
  Dim x as Single
  Dim p as Integer
  p = 345
  x = MAPPRO1.Int2Lat(p)
  Text1 = Str(x)
End Sub
```

Delphi Example

```
Procedure TForm1.Button3Click(Sender: TObject);
```

```
var Yd:string[12];
    p:integer;
begin
  p:=345;
  floattstr(MapPro1.Int2Lat(p));
  panel2.caption:='Lat = '+Yd;
end;
```

Int2Lon(i:Integer):Double — Function

Returns a real value of the Longitude, given the X coordinate in screen units.

Note: An incorrect coordinate will be returned if the map has been rotated.

VB Example

```
Private Sub Command39_Click()
  Dim x as Single
  Dim p as Integer
  p = 345
  x = MAPPRO1.Int2Lon(p)
  Text1 = Str(x)
End Sub
```

Delphi Example

```
Procedure TForm1.Button6Click(Sender: TObject);
var Yd:string[12];
     p:integer;
begin
  p:=412;
  floattstr(MapPro1.Int2Lon(p));
  panel2.caption:='Lon = '+Yd;
end;
```

IsDrawing:Boolean — Property

When true, it indicates that the control is in the process of drawing the map. The *true* state of the Boolean initiates when the Windows OS message to Paint is issued, and is terminated when all painting of the control is completed.

VB Example

```
Private Sub Command39_Click()
  If Mappro1.IsDrawing = False Then
  MapPro1.ZoomAll
End Sub
```

Delphi Example

```
Procedure TForm1.Button8Click(Sender: TObject);
begin
  with MapPro1 do
  begin
    If Not(IsDrawing) then ZoomPan(2);
  end;
end;
```

ItemFontSize:Integer — Property

Sets the size of the font to be used for the user items placed on the map control by the SetItem method.

VB Example

```
Private Sub Command105_Click()
 ' Set font size for labeling user items
 MapPro1.ItemFontSize = 20
 ' Set a user item
 MapPro1.SetItem 1, -70, 40
 MapPro1.SetItemString 1, "Sample"
 ' Refresh the screen to show the label
 MapPro1.Refresh
End Sub
```

Kms Property:Double

This property sets the scale of the map control. The value of Kms is the number of Kilometers per logical inch of the control. This results in much more accurate zooming than that using the Magnitude property.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Kms=27
End Sub
```

Delphi Example

```
Procedure TForm1.Button1Click(Sender: TObject);
begin
  { Set Scale to 20 Kms to the inch}
    MapPro1.Kms=27;
end;
```

Lat2Int(x:OLEVariant):Integer — Function

Returns an integer giving the Latitude equivalent in device coordinates. Truncates to the nearest pixel.

Note: An incorrect value will be returned if the map has been rotated. See the LL2INT function for getting the equivalent information, even with rotated maps.

VB Example

```
Private Sub Command39_Click()
  Dim x as Integer
  x = MapPro1.Lat2Int(MapPro1.ycord)
  Text1 = Str(x)
End Sub
```

Delphi Example

```
Procedure TForm1.Button3Click(Sender: TObject);
var Yd:string[12];
  begin
  inttstr(Lat2Int(MapPro1.ycord),Yd);
  panel2.caption:='Lon Equivalent: '+Yd;
end;
```

LatBottom:Double — Property

Indicates the Latitude of the lower edge of the map view window.

VB Example

```
Private Sub Command39_Click()
  Dim x as Double
  x = MapPro1.LatBottom
End Sub
```

Delphi Example

```
Procedure TForm1.Button11Click(Sender: TObject);
var xt:double;
begin
  xt:=MapPro1.Latbottom;
end;
```

LatCenter:Double — Property

Indicates the Latitude of the center point of the map view window.

VB Example

```
Private Sub Command105_Click()
Dim x As Double, y As Double, xd As Double, yd As Double
Dim deltaX As Double, deltaY As Double

' Obtain center coordinates
 x = MapPro1.LonCenter
 y = MapPro1.LatCenter
 'Calculate center coordinates using alternate method
 yd = (MapPro1.LatTop + MapPro1.LatBottom) / 2
 xd = (MapPro1.LonLeft + MapPro1.LonRight) / 2
 ' Deltas shoud be zero
 deltaX = x - xd
 deltaY = y - yd
End Sub
```

LatTop:Double — Property

Indicates the Latitude of the upper edge of the map view window.

VB Example

```
Private Sub Command39_Click()
  Dim x as Double
  x = MapPro1.LatTop
End Sub
```

Delphi Example

```
Procedure TForm1.Button15Click(Sender: TObject);
var x:double;
begin
  x:=MapPro1.LatTop;
end;
```

LL2INT(x,y:OLEVariant):LongInt — Function

Take the lon (x) and lat (y) coordinates and return a packed long integer containing the rotated screen coordinates, lon (x) in the 'high' word and lat (y) in the 'low' word.

Note: The user does not have to perform the calculation transformation. For example, if the user was drawing a line from unrotated coordinates x1,y1 to x2,y2, then in Delphi,

```
xy1:=MapPro1.LL2INT(x1,y2);
xx1:=loword(xy1);
yy1:=hiword(xy1);
xy2:=MapPro1.LL2INT(x2,y2);
xx2:=loword(xy2);
yy2:=hiword(xy2);
```

xx1,yy1 and xx2,yy2 would be the rotated screen coordinates corresponding to x1,y1 and x2,y2.

VB Example

```
Private Sub Command39_Click()
  Dim x as Integer
  x = MapPro1.LL2Int(MapPro1.ycord)
  Text1 = Str(x)
End Sub
```

Delphi Example

```
Procedure TForm1.MapPro1paintAfter(Sender: TObject; dc: Integer);
var testing1,testing2,i,j,ix1,iy1,ix2,iy2:longint;
    hp:hpen;
    DInteger:integer;
begin
  i := MapPro1.LL2int(MapPro1.LonLeft, MapPro1.LatTop);
  j := MapPro1.LL2int(MapPro1.LonRight, MapPro1.LatBottom);
  iX1 := i div 65536;
  iY1 := i Mod 65536;
  iX2 := j div 65536;
  iY2 := j Mod 65536;
  val(edit2.text,MyBlue,code);
  hp:=createpen(ps_solid,5,RGB(MyRed,MyGreen,MyBlue));
  hp:=selectobject(dc,hp);
  MoveToEx(dc, iX1, iY1, nil);
  LineTo(dc, iX2, iY2);
  deleteobject(selectobject(dc,hp));
end;
```

LLMode:Integer — Property

Sets the format to be used when displaying the coordinates in the toolbar area. Also sets the format for the Grid line labels.

LLMode can take the following values:

0 - LLDMS (Display format DD MM' SS")
1 - LLDEG (Display format DD.xxxxxx)
2 - LLInternal (Display internal coordinate, LongInt)

3 - LLScreen (Display screen coordinates)

Note: The user may also cycle through these different display formats by clicking in the coordinate display area when the toolbar is visible.

Delphi Example
```
Procedure TForm1.Button15Click(Sender: TObject);
var x:double;
begin
  x:=MapPro1.LLMode:=1;
end;
```

Load_Air:Boolean — Property

When true enables loading of the airport data layer. The layer remains visible all the way down to the detailed Tiger data level (<2 mi).

VB Example
```
Private Sub Command39_Click()
  MapPro1.Load_Air = Not MapPro1.Load_Air
End Sub
```

Delphi Example
```
Procedure TForm1.Button15Click(Sender: TObject);
begin
  MapPro1.Load_Air:=not(MapPro1.Load_Air);
end;
```

Load_City:Boolean — Property

When true enables loading of the city, place and county names data layer. It remains visible all the way down to the detailed Tiger data level (<2 mi).

VB Example
```
Private Sub Command39_Click()
  MapPro1.Load_city = Not MapPro1.Load_city
End Sub
```

Delphi Example
```
Procedure TForm1.Button15Click(Sender: TObject);
begin
  MapPro1.Load_City:=not(MapPro1.Load_City);
end;
```

Load_County:Boolean — Property

When true enables loading of county area information so that it may be shaded. Visibility of this layer is turned off at the Tiger data level (<2 mi).

VB Example

```
Private Sub Command39_Click()
  MapPro1.Load_county = Not MapPro1.Load_county
End Sub
```

Delphi Example

```
Procedure TForm1.Button5Click(Sender: TObject);
begin
  MapPro1.Load_county:=not(MapPro1.Load_county);
end;
```

Load_Highway:Boolean — Property

When true, it enables loading of the National Highway Network data layer. Visibility of this layer is turned off at the Tiger data level (<2 mi).

VB Example

```
Private Sub Command39_Click()
  MapPro1.Load_highway = Not MapPro1.Load_highway
End Sub
```

Delphi

```
Procedure TForm1.Button9Click(Sender: TObject);
begin
  MapPro1.Load_highway:=not(MapPro1.Load_highway);
end;
```

Load_Hydro:Boolean — Property

When true, it enables loading of the USGS major water bodies and rivers layer. Visibility of this layer is turned off at the Tiger data level (<2 mi).

VB Example

```
Private Sub Command39_Click()
  MapPro1.Load_hydro = Not MapPro1.Load_hydro
End Sub
```

Load_Landmark:Boolean — Property

When true, it enables loading of the landmark data layer. Visibility of this layer is enabled only below the Tiger data threshold (<2 mi).

Note: Depending on the map scale, landmarks are displayed as,
(a) 2x2 bit color coded marker, at about 2 mi scale

(b) 12x12 bit color coded rectangle with a landmark number, at about 0.5 mi scale (see Precision Mapping Help for landmark identification numbers)

(c) 12x12 bit color icon and the actual landmark name (where available) at less than 1000 ft scale (see Precision Mapping Help for landmark identification icons and numbers)

VB Example
```
Private Sub Command39_Click()
  MapPro1.Load_landmark = Not MapPro1.Load_landmark
End Sub
```

Delphi
```
Procedure TForm1.Button7Click(Sender: TObject);
begin
  MapPro1.Load_landmark:=not(MapPro1.Load_landmark);
end;
```

Load_Park:Boolean — Property

When true enables loading of the parks data from the mparks.zpx file.

VB Example
```
Private Sub Command39_Click()
  MapPro1.Load_state = Not MapPro1.Load_state
End Sub
```

Delphi
```
Procedure TForm1.Button5Click(Sender: TObject);
begin
  MapPro1.Load_state:=not(MapPro1.Load_state);
end;
```

Load_Shore:Boolean — Property

When true enables loading of the shoreline data from the mshore.zpx file in the DATA1 folder. Note that this is the shoreline data that becomes visible below the 200 mile threshold. It has no effect on the course world outline that is bound to the OCX.

Load_State:Boolean — Property

When true enables loading of the state and county political border data.

VB Example
```
Private Sub Command39_Click()
  MapPro1.Load_state = Not MapPro1.Load_state
End Sub
```

Delphi
```
Procedure TForm1.Button5Click(Sender: TObject);
begin
  MapPro1.Load_state:=not(MapPro1.Load_state);
end;
```

Load_World:Boolean — Property

When true enables loading of the world reference data from the mworld.zpx file in the DATA1 folder. Note that this is the shaded country polygon data that becomes visible below the 200 mile threshold. It has no effect on the course world outline that is bound to the OCX.

LoadConfig(path:String) — Procedure

Loads the configuration file specified by the full path specification, 'Path'. Note that the default configuration file name, used by AutoConfig is **MapPro71.cfg**

VB Example

```
Private Sub Command39_Click()
  MapPro1.Loadconfig (App.Path & "\MapPro.cfg")
End Sub
```

Delphi

```
Procedure TForm1.Button1Click(Sender: TObject);
begin
  // Load the file MAPPRO.OCX from the directory d:\MyData
  MapPro1.LoadConfig('d:\MyData\MAPPRO.CFG);
end;
```

Loaded:Boolean — Method

This is used when running the OCX in a non-visual environment. This forces the OCX to load all of the necessary data files and configurations that are normally done in a visual environment automatically.

VB Example

```
Private Sub Form_Load()

    'Create the object
    Set MapPro1 = CreateObject("MapPro50.Pmap")

    'Load in all of the little things that are in the
    'Config file - if you want to change your paths
    'do it after the Loaded statement
    MapPro1.AutoConfig = True
    MapPro1.Loaded

    'Set your map size
    MapPro1.ResizeCtl 300, 300

    'Echo your paths so you know it loaded the config
    'file.  Sometimes you don't know which MapPro50.cfg
    'file it is loading in.
    MsgBox MapPro1.Path_states0

End Sub

Private Sub Command1_Click()
    MapPro1.Miles = 1
    MapPro1.GotoPoint -85, 41
    MapPro1.SetItem 1, -85, 41
    MapPro1.SetItemString 1, "MapOCX"
    MapPro1.SaveToGif "C:\Test.gif"
    Picture1.Picture = LoadPicture("C:\Test.gif")
    Picture1.Refresh
End Sub
```

LoadExclusion(filename:String) — Procedure

Loads the list of bounding box coordinates from the specified file. The exclusion bounding polygons read from the file replace any such polygons currently in the memory-based exclusion list. The bounding boxes are used to exclude areas in route.

Delphi

```
Procedure TForm1.Button35Click(Sender: TObject);
begin
  // Load exclusions file - assumed extension: .exl
  MapPro1.LoadExclusion('MyRtExcFile');
end;
```

LoadStreets(s:String) — Procedure

Loads a User Street file specified by a file name. It replaces any user streets currently in memory.

VB Example

```
Private Sub Command2_Click()
    MapPro1.LoadStreets ("C:\Steets.Str")
End Sub
```

LoadViaFile(s:String) — Procedure

Loads a file containg Via points to be used for routing.

VB Example

```
Private Sub Command2_Click()
    Mappro1.LoadStreets ("C:\Steets.via")
End Sub
```

Lon2Int(x:OLEVariant):Integer — Function

Returns an integer giving the Longitude equivalent in device coordinates. Truncates to the nearest pixel.

Note: An incorrect value will be returned if the map has been rotated. See the LL2INT function for getting the equivalent information, even with rotated maps.

VB Example

```
Private Sub Command39_Click()
  Dim x as Integer
  x = MapPro1.Lon2Int(MapPro1.xcord)
  Text1 = Str(x)
End Sub
```

Delphi

```
Procedure TForm1.Button4Click(Sender: TObject);
var Xd:string[12];
begin
```

LonCenter:Double — Property

Indicates the Longitude of the center point of the map view window. (Also see LatCenter)

LonLatStr(x,y:OLEVariant):String — Function

Returns a formatted string for the specified Lon/Lat coordinates. X and Y are decimal coordinates, and the returned string contains the coordinates in Deg.Min.Sec format.

VB Example

```
Private Sub Command39_Click()
  Text1.text = MapPro1.lonlatstr(-120,32)
End Sub
```

Delphi

```
Procedure TForm1.Button7Click(Sender: TObject);
begin
  panel2.caption:=MapPro1.lonlatstr(-120,32);
end;
```

LonLeft:Double — Property

Indicates the Longitude of the left edge of the map view window.

VB Example

```
Private Sub Command39_Click()
  Dim x as Double
  x = MapPro1.LonLeft
End Sub
```

Delphi

```
Procedure TForm1.Button5Click(Sender: TObject);
var xl:double;
begin
  xl:=MapPro1.LonLeft;
end;
```

LonRight — Property

Indicates the Longitude of the right edge of the map view window.

VB Example

```
Private Sub Command39_Click()
  Dim x as Double
  x = MapPro1.LonRight
End Sub
```

Delphi

```
Procedure TForm1.Button18Click(Sender: TObject);
{Calculate screen center coordinates }
var Centerx,Centery:double;
begin
  CenterX:=(MapPro1.lonleft+MapPro1.lonright)/2;
  CenterY:=(MapPro1.lattop+MapPro1.Latbottom)/2;
end;
```

Magnitude:Integer — Property

This property controls the scale of the map in the viewport. When the magnitude property is set, the value of the internal "Scale" property changes as well.

Magnitude is an enumerated variable that can take the following values (Note that the scale of Magnitude is approximate. See the Miles property for more accurate scaling):

```
 0 : M500_0   500 mi to the inch
 1 : M300_0   300 mi to the inch
 2 : M200_0   200 mi to the inch
 3 : M100_0   100 mi to the inch
 4 : M050_0    50 mi to the inch
 5 : M030_0    30 mi to the inch
 6 : M020_0    20 mi to the inch
 7 : M010_0    10 mi to the inch
 8 : M005_0     5 mi to the inch
 9 : M003_0     3 mi to the inch
10 : M002_0     2 mi to the inch
11 : M001_0     1 mi to the inch
12 : M000_5   0.5 mi to the inch
13 : M000_3   0.3 mi to the inch
14 : M000_2   0.2 mi to the inch
15 : M000_1   0.1 mi to the inch
```

Note: Although the largest enumerated magnitude (15) corresponds to approximately 0.1 mi/inch, internally, the maximum scale limit is 0.025 mi/inch. So, setting a magnitude of 15 would zoom in, however, further zooming might be possible using the mouse. Also, setting the Scale property to a value greater than 10, would result in maximum magnification.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Magnitude = 3
End Sub
```

Delphi

```
Procedure TForm1.Button2Click(Sender: TObject);
begin
  { Set Magnitude scale to 50 mi/in }
    MapPro1.Magnitude:=4;
end;
```

MainLay:Boolean — Property

When true enables loading of the Main Tiger Data Set.

VB Example

```
Private Sub Command39_Click()
  MapPro1.MainLay = False
```

```
End Sub
```

Delphi

```
Procedure TForm1.Button3Click(Sender: TObject);
begin
  MapPro1.MainLay:=False;
end;
```

MapCount:LongInt — Property

This read-only property returns the number of map generations or map redraw actions. This is used when licensing for internet server or per transaction licensing.

Mapmode:Integer — Property

This property controls the built in behavior of zooming. Autozooming is enabled when mapmode is set to a value of 0 (MdZoom) and is evidenced by the Zoom circle cursor. A value of 1 (MdUser) permits users to set their own cursor type, and it disables the default zoom/window mode. Note that while in this latter mode, the user still has access to the Xcord, Ycord properties which, for example, may be used with the SetItem procedures to place user-created objects, using an OnMouse... event. A value of 2 permits users to add/delete or edit user-defined street segments. See APPENDIX L for Mode 2 (Adding Streets).

Note: When MapMode=2 is invoked, the cursor changes to a plain cross-hair that can be used to locate vertices of a new street segment. As the cursor moves around, a circle of "snap" or "attach" influence can be seen tracking the cursor movement always being on an existing road segment. If the cursor is inside this influence circle, when the left mouse button is clicked, then the current vertice will attach (snap) to the existing road point in the circle.

The user may continue to press the left mouse button and define new vertices (belonging to the same street polyline) at will. When the desired number of vertices have been defined, the user may press the right mouse button to signify completion of the current street polyline definition. It should be noted that when the Street editing mode is invoked, all user-defined road segments become cyan for better/quicker identification.

When the right mouse button is pressed, a dialog appears that permits the user to specify the name for the created segment, as well as to assign the desired road attribute. The options available in this dialog are:

(File) New Clears all currently defined segments from memory (make certain you have saved any road segments you want, prior to selecting this command).

(File) Open Load a user specified external roads file, (see further down for file structure). Note that this operation will erase ALL user-defined segments currently in memory before loading the specified roads file (also see File, Merge.)

(File) Save The user may save the currently defined street segments to a file (the extension .STR is automatically appended)

(File) Merge Load a user specified external roads file, (see further down for file structure). Note that this operation will does NOT erase-user defined segments currently in memory but merges them with the ones loaded from the specified file.

(File) Exit Close the street editing dialog (note that this does NOT cancel the street editing mode, which can only be done by setting the appropriate MapMode value.)

(Options) ZoomAll Zooms the view port to the extents of all User Defined streets currently in memory.

(Options) Attach Toggles the display and operation of the "attach" circle on or off.

(Options) Ortho When toggled on, only horizontal and vertical street orientations are permitted.

Name This is the name assigned to the current road segment by the user. It's used to label the road segment, search for it, etc. (Note: When searching for streets, these road segments are identified as "User Defined" in the listbox that appears in the search dialog.)

Road Type Five Road types are allowed. The descriptions of these types in the dialog are self-explanatory.

Add [Button] Adds a newly defined road segment to the list of road segments already in memory. Note that these segments are NOT saved unless the File, Save command (from this dialog) is executed.

Modify [Button] Replaces the attributes of the currently selected road segment with new ones specified in the dialog.

Delete [Button] Deletes the currently selected road segment from the list of segments in memory (but not from a file such segments have been saved
in, unless the File, Save command is executed subsequent to the deletion)

Cancel [Button] Close the street editing dialog (note that this does NOT cancel the street editing mode, which can only be done by setting the appropriate MapMode value.)

a) A defined road segment is selected for editing, or deletion, by placing the cursor on the segment and pressing the left mouse button while holding down the Shift key. The selected road segment will assume flashing highlight attribute to clearly show the used that it is being modified

b) The user-created road segments are NOT visible at scales above the Tiger Street level scale, i.e. about 2 miles. Also, the editing mode ahould not be activated if the current scale is not at the Tiger street level scale or lower.

c) The number of road segments that can be loaded at any given time is 50,000.

d) User defined road segments may be "searched for" using the standard searching techniques of MAPPRO, by specifying the assigned street name.

e) While in the street editing mode, road segment vertices may be moved by placing the mouse cursor on them, and holding down the control key and the left mouse button. This action engages the vertex which may them be dynamically moved to a new location. When the mouse button is released, the new location of the vertex become permanent.

User-Created Street file format (plain text)

```
Street File:Chicago Map Corporation
STR "RoadName/SecondaryFileName" Class N
x1 y1
x2 y2
...
...
xN yN
```

where: *First Line is an identifying header line*

STR - Keyword used internally

RoadName - Name specified for the road by the user. Used for display and search purposes. Note that a secondary name may also be specified using the slash character as a separator.

Class - 50: Interstate
59: Primary Highway
68: Major Road
77: Minor Road
93: Ramp

N - Number of points for this Road segment

x1, y1 - The longitude and latitude (x and y) coordinates for each of the segments defining this road (in decimal degree units). It should be noted that streets files created with earlier releases of MAPPRO40.OCX, i.e., without the header line and using internal coordinates, may still be read by the OCX transparently.

VB Example

```
Private Sub Command39_Click()
    MapPro1.mapmode = MdUser
    MapPro1.cursor = 17
End Sub
```

Delphi

```
Procedure TForm1.SpeedButton10Click(Sender: TObject);
begin
  { set default mode }
  MapPro1.Mapmode:=mdZoom;
end;

Procedure TForm1.Button4Click(Sender: TObject);
begin
  { Set User mode and North/South cursor }
  MapPro1.mapmode:=MdUser;
```

```
  MapPro1.cursor:=crsizens;
end;
```

MapUnits:Integer — Property

This property controls the units used in the mapping application of the OCX control. If equal to 0, or 'Mumi', then the units are miles. If equal to 1, or 'Mukm' the units are kilometers.

VB Example

```
Private Sub Command39_Click()
  MapPro1.mapmode = MdUser
End Sub
```

Delphi

```
Procedure TForm1.Button4Click(Sender: TObject);
begin
  { Set Units to miles }
  MapPro1.mapmode:=MdUser;
end;
```

Miles:Double — Property

This property sets the scale of the map control. The value of Miles is the number of miles per logical inch of the control. This results in much more accurate zooming than that using the Magnitude property.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Miles = 27
End Sub
```

Delphi

```
Procedure TForm1.Button1Click(Sender: TObject);
begin
  { Set Scale to 27 miles to the inch}
  MapPro1.Miles=27;
end;
```

OnClick — Event

Issued when a mouse clicks on the control surface.

VB Example

```
Private Sub MapPro1_Click()
  Dim x As Integer
  x = MapPro1.Lat2Int(MapPro1.Ycord)
  Text1 = Str(x)
End Sub
```

Delphi

```
Procedure TForm1.MapPro1Click(Sender: TObject);
{-------------------------------------------------}
{ In Click mode.  Select current location         }
```

```
{-------------------------------------------------}
var dc:integer;
begin
  defx1:=MapPro1.xcord;
  defy1:=MapPro1.Ycord;
  str(defx1:10:6,xc1);
  str(defy1:10:6,yc1);
  Panel2.Caption:='Selected: Point #1 = '+ xc1 + ', ' + yc1);
end;
```

OnDblClick — Event

Issued when a mouse double clicks on the control surface.

VB Example

```
Private Sub MapPro1_DblClick()
  Dim x As Integer
  x = MapPro1.Lat2Int(MapPro1.Ycord)
  Text1 = Str(x)
End Sub
```

Delphi

```
Procedure TForm1.ListBox1DblClick(Sender: TObject);
var temp:string;
    dc,j,i,code:integer;
    x,x1,y1,x2,y2:real;
begin
  Temp:=listbox1.items[listbox1.itemindex];
  j:=pos(chr(9),temp);
  temp:=copy(temp,j+1,length(temp)-j);
  j:=pos(chr(9),temp);
  val(copy(temp,1,j-1),x1,code);
  temp:=copy(temp,j+1,length(temp)-j);
  j:=pos(chr(9),temp);
  val(copy(temp,1,j-1),y1,code);
  temp:=copy(temp,j+1,length(temp)-j);
  MapPro1.gotopoint(x1,y1);
end;
```

OnDirect — Event

Issued when DirectDraw has finished processing the Map, the Overlay and the User layer, but prior to setting the scale factor used by DirectDraw to map the OCX control surface to the user specified DC.

VB Example

```
Private Sub MapPro1_Direct(ByVal dc As Long)
  Call MapPro1.DrawScalebar(Printer.hdc, 3, 3)
End Sub
```

Delphi

```
Procedure TForm1.MapPro1Direct(Sender: TObject; dc: Integer);
{-----------------------------------------------}
{ Processes to be done following a DirectDraw   }
{-----------------------------------------------}
var hp:hpen;
```

```
begin
  hp:=createpen(ps_solid,4,RGB(0,0,255));
  hp:=selectobject(dc,hp);
  {Paint Blue line}
  MoveToEx(dc, 10,10, nil);
  Lineto(dc,60,60);
  deleteobject(selectobject(dc,hp));
end;
```

OnDirectBefore — Event

Performs the specified Operations immediately before re-painting the map by DirectDraw on the user specified surface. The painting on the control surface is completed in this order:

- The OnDirectBefore Event is triggered
- The map is painted
- The overlay is painted
- The OnDirect event is triggered
- The user layer is painted

VB Example

```
Private Sub MapPro1_PaintBefore(ByVal dc As Long)
  Call MapPro1.DrawScalebar(Printer.hdc, 3, 3)
End Sub
```

Delphi

```
Procedure TForm1.MapPro1directBefore(Sender: TObject;dc: Integer);
{-----------------------------------------------------------}
{ Draw a blue line at the specified Lat/Lon coordinates    }
{ prior to the map being drawn to the dc.                  }
{ Note that in order for the line draw here to be visible, }
{ shading has to be turned off                             }
{-----------------------------------------------------------}
begin
  {Use DrawLine so that the line can be clipped to the
   viewing window if necessary }
  MapPro1.Drawline(dc,-101.34056,34.55798,
  -99.32330,30.676885,4,clblue,R2_CopyPen);
end;
```

OneWayColor:Integer — Property

Color to be used for highlighting One-way street segments above the double-line-road zoom levels, and the arrows when at double-line level. A negative value indicates no highlighting will be done. Note that because of the need to retain the data structure in older versions of MapPro, only colors primary 0 - 16 may be assigned to this property.

Delphi

```
Procedure TForm1.Button29Click(Sender: TObject);
// Cycle through the 16 colors to highlight one way roads
begin
  MapPro1.OneWayColor:=MapPro1.OneWayColor+1;
  // Make sure it's limited to 16
```

```
  if MapPro1.Onewaycolor>16 then MapPro1.onewaycolor:=0;
  MapPro1.redraw;
end;
```

OneWayShow — Property

Enables the automatic drawing of arrows for road segments that are one way, when the view port is zoomed to the double-line road level, or the different color lines when zoomed out.

Delphi

```
Procedure TForm1.Button30Click(Sender: TObject);
// Toggle the onewayshow flag
begin
  with MapPro1 do
  begin
    MapPro1.OneWayShow:=not(OneWayShow);
    if Onewayshow=true then button30.caption:='s:T'
         else button30.caption:='s:F';
    MapPro1.Redraw;
  end;
end;
```

OneWayUse — Property

Enables the use of the one-way information when calculating a route. The default is true. Note that is also disables the old logic (used with TIGER data), where the angle of attack was used to determine the appropriateness of using a limited access highway ramp (that was the only way On/Off ramps could be discerned in the past. This was introduced with the use of GDT/TeleAtlas data.

Delphi

```
procedure TForm1.Button33Click(Sender: TObject);
// Toggle the onewayuse flag
begin
  with MapPro1 do
  begin
    OneWayUse:=not(OneWayUse);
    if OneWayUse=true then button33.caption:='u:T'
      else button33.caption:='u:F'
  end;
end;
```

onCADChange(Current:Long) — Event

```
This event is fired each time the ObjectType is changed, and it returns the
current ObjectType.  The valid ObjectTypes, corresponding to valid CAD objects,
are shown below.  If ObjectType=-1, then the control is in object selection
```

mode. If ObjectType=0, then the control is in NUL mode, i.e., the mapMode may be set to mbZoom.

#	**Object Type**
1	Line
2	Rectangle
3	Ellipse
4	Polyline
5	Polygon
6	Marker
7	Text
8	Circle
9	Regular Polygon
10	Free Hand (*)
11	Arrow
12	Bezier
13	Symbol
14	Text Bubble
15	Grouped Object
13	MetaObj

VB Example

```
Private Sub MapPro1_CadChange(ByVal Current As Long)
' Check to see if Current (ObjectType) is -1 and change MapMode
  If Current = -1 Then MapPro1.MapMode = MdZoom
  Beep
End Sub
```

OnFind — Event

This event is triggered by the FindStreet method when a street matching the search specification has been found. The user should then query the Street property and parse the information of interest from the resulting string.

VB Example

```
Private Sub MapPro1_Find()
  Text1 = MapPro1.Street
End Sub
```

Delphi

```
Procedure TForm1.MapPro1Find(Sender: TObject);
begin
  If ProcMode=4 then listbox1.items.add(MapPro1.street);
  {Display number of streets found}
  panel4.caption:=' Found: '+inttostr(listbox1.items.count);
  Application.ProcessMessages;
end;
```

OnFindDir — Event

Road segments found when the FindRoute method, or the ExecRoute dialog are executed, are combined into explicit directions. For each explicit direction placed in the ExecRoute listbox, the OnfindDir is fired, and a string is passed in it. The string contains the following information, separated by tabs (#9 character):

```
Road Name;X,Y coordinates;Instruction;Direction;Distance;Time
```

VB Example

```
Private Sub MapPro1_FindDir(ByVal s As String)
  List1.AddItem s
End Sub
```

Delphi

```
Procedure TForm1.MapPro1FindDir(Sender: TObject;s:string);
begin
  listbox2.items.add(s);
end;
```

OnFindPlace — Event

For each place that is found, when the user calls FindCity, an OnFindPlace event is triggered. The user must then query the 'Result' string property in the OnFindPlace event handler, and store it or otherwise process the string, e.g., add it to a list box, parse it to its appropriate components. The string stored in "Result" contains the following information separated by a tab character (#9).

Name - The place (City, Town, etc.) name from the places database
State - The two letter state abbreviation
Lon - Longitude (from the places database)
Lat - Latitude (from the places database)

FindCity supports two different paradigms:

(a) If an explicit string is searched for, then the OnFindPlace is triggered and a single hit (if found) returns in the "Result" property of the control.

(b) If a wildcard search is specified, then an OnFind event is triggerred and found hits are returned in the "Street" property.

VB Example

```
Private Sub MapPro1_FindPlace()
  List1.AddItem MapPro1.Result
End Sub
```

```
Example of how this works:

** Explicit Search **
     FindCity Search String: Lemont
           Event Triggerred: OnFindPlace
    Number of hits Returned: 1
```

Delphi Example
```
Procedure TForm1.MapPro1FindPlace(Sender: TObject);
begin
  listbox1.items.add(MapPro1.result);
  {Increase the count and add item to array}
  inc(m);
  res[m]:=MapPro1.result;
  {Display the # of places found}
  panel4.caption:=' Found: '+inttostr(listbox1.items.count);
  Application.ProcessMessages;
end;
```

OnFindRte — Event

For each road segment found when the FindRoute method is used, an OnFindRte event is triggered. The user must then query the 'Result' property in the OnFindRte event handler, and store it or otherwise process its value, e.g., add it to a list box, parse

Note: For details on what the 'Result' string contains, see the documentation for the FindRoute method.

VB Example
```
Private Sub MapPro1_FindPlace()
  List1.AddItem MapPro1.Result
End Sub
```

Delphi
```
Procedure TForm1.MapPro1FindRte(Sender: TObject);
begin
  listbox2.items.add(MapPro1.result);
end;
```

OnMouseDown — Event

Issued when the mouse button is held down on the control surface.

VB Example
```
Private Sub MapPro1_MouseDown(button As Long, shift As Long, x As
Long, y As Long)
  Dim dc%
  Dim di%
  dc = GetDC(MapPro1.Handle)
  MapPro1.DrawObject dc, MapPro1.Xcord, MapPro1.Ycord, 3, vbRed
  di = ReleaseDC(MapPro1.Handle, dc)
End Sub
```

OnMouseMove — Event

Issued when the mouse moves over the control surface.

VB Example
```
Private Sub MapPro1_MouseMove(button As Long, shift As Long, x As
Long, y As Long)
  Form1.Caption = "lat: " & MAPPRO1.Ycord & " Lon:" &
MAPPRO1.Xcord
```

```
End Sub
```

Delphi

```
Procedure TForm1.MapPro1MouseMove(Sender: TObject;
                  Shift: TShiftState; X, Y: Integer);
var xtemp,ytemp,x2,y2:string[12];
begin
  with MapPro1 do
  begin
    str(xcord:10:6,xtemp);
    str(ycord:10:6,ytemp);

    { Now, let's get the coordinates in Deg.Min.Sec form as
      well, and then place both types on panel2 }
    panel2.caption:='W:'+xtemp+'  N:'+ ytemp
               +'('+DMS(xcord)+', '+DMS(ycord)+')';
  end;
end;
```

OnMouseUp — Event

Issued when the mouse button is released.

VB Example

```
Private Sub MapPro1_MouseUp(button As Long, shift As Long, x As
Long, y As Long)
  If MapPro1.Mapmode = MdUser And button = 1 Then
   MapPro1.SetItem 1, MapPro1.Xcord, MapPro1.Ycord
   Call OlePmMap.SetItemlocalbitmap(1, App.Path & "\" & "car.bmp")
   MapPro1.SetItemString 1, "My car"
   MapPro1.Redraw
  End If
End Sub
```

Delphi

```
Procedure TForm1.MapPro1MouseUp(Sender: TObject; Button:
     TMouseButton; Shift: TShiftState; X, Y: Integer);
{-------------------------------------------------------}
{ Simply displays a message indicating the mouse button  }
{ has been Released. Better illustrated if the right     }
{ mouse button is pressed and released                   }
{-------------------------------------------------------}
begin
  Panel2.caption:='Mouse Button is Up (Released)';
  { annunciate the buttonup event}
  messagebeep(0);
  Application.ProcessMessages;
end;
```

OnOptiRouterMsg **Event**

This event is fired at various points during the Optimized Route calculation, following an OptiRouter.Calculate call, or the selection the option to perform the calculations from the OptiRouter dialog. Code is reserved, and not used at this time. MsgStr contains one of the following strings identified what part of the calculation the OptiRouter is in.

- **Loading NetWork** – The OptiRouter has started the process of loading the Road Node Network which will be used for the routing calculations.
- **A series of data grid ID's** specifying which grids are currently loaded.
- **Network Loaded** – The OptiRouter has successfully loaded the Road Nodes Network
- **Finished Opti Route xxx mi** – The OptiRouter has finished its calculations, the total travel distance in miles is alco displayed.

VB Example

```
Private Sub MapPro1_OptiRouteMsg(ByVal Code As Long,
ByVal MsgStr As String)
  ' Display the Status message returned each time the
  ' event is fired, in a list box
  List1.AddItem "Event fired: " & MsgStr
End Sub
```

OnPaintAfter **Event**

Performs the specified operations immediately after re-painting the map on the surface of the OCX control. This takes place immediately after the map is re-painted, but prior to the user layer being painted on the control surface. The painting on the control surface is completed in this order:

- The underlay is painted
- The map is painted
- The overlay is painted
- The OnPaintAfter event is triggered
- The user layer is painted

VB Example

```
Private Sub MapPro1_PaintAfter(ByVal dc As Long)
  Call MapPro1.DrawScalebar(dc, 5, 8)
End Sub
```

Delphi

```
Procedure TForm1.MapPro1paintAfter(Sender: TObject; dc: Integer);
begin
  {Draws a scale bar and a text bubble on top of the map surface}
  MapPro1.drawscalebar(dc,5,5);
  MapPro1.Drawbubble(dc,-125,45,'Sample');
end;

procedure TForm1.MapPro1paintAfter(Sender: TObject; dc: Integer);
var hp:hpen;
begin
  With MapPro1 do
  begin
    {Get Device Context and select Pen}
    hp:=createpen(0,1,RGB(0,0,$ff));
```

```
    hp:=selectobject(dc,hp);
    {Draw A line from Denver to Oklahoma City}
    movetoex(dc,Lon2Int(-104.83),Lat2Int(39.64),nil);
    lineto(dc,Lon2Int(-97.35),Lat2Int(35.42));
    {Deselect Pen and release device context}
    deleteobject(selectobject(dc,hp));
  end;
end;
```

OnPmapEvent — Event

This event is fired as needed by the user. See FirePMapEvent.

Delphi Example

```
Procedure TForm1.Button6Click(Sender: TObject);
// Fire an event with a 500 msec delay and increment the eventide
#
begin
  inc(eid);
  MapPro1.FirePmapEvent(eid,500);
end;

procedure TForm1.MapPro1PmapEvent(Sender: TObject; EventID:
Integer);
// When the event is fired, it is trapped here an a string is
added
// to the listbox to let the user know
begin
  ListBox1.items.add('Fired Event #'+inttostr(Eventid));
end;
```

OnPaintBefore — Event

Performs the specified operations prior to re-painting the map on the surface of the OCX control.

VB Example

```
Private Sub MapPro1_PaintBefore(ByVal dc As Long)
  MapPro1.DrawLine dc, -89.76, 41.12, -88.99, 42.43, 4, vbGreen,
13
End Sub
```

Delphi

```
Procedure TForm1.MapPro1PaintBefore(Sender: TObject; dc:
     Integer);
{-------------------------------------------------}
{ Draw Thick Red line before drawing the map      }
{ It's visible ONLY outside the USA boundary      }
{ Unless shading is turned OFF                    }
{-------------------------------------------------}
begin
  MapPro1.Drawline(dc,-115.34056,28.55798,
                      -90.32330,49.676885,8,
                          clRed,R2_CopyPen);
end;
```

OnResize — Event

This event is triggered when the height or width of the control is changed.

VB Example

```
Private Sub MapPro1_Resize()
  Dim dc%
  Dim di%
  dc = GetDC(MapPro1.Handle)
  MapPro1.DrawScalebar dc, 5, 8
  di = ReleaseDC(MapPro1.Handle, dc)
End Sub
```

Delphi

```
Procedure TForm1.MapPro1Resize(Sender: TObject);
var dc:integer;
begin
  dc:=getdc(MapPro1.handle);
  DrawObject(dc,MapPro1.xcord,MapPro1.ycord,2,clRed);
  releasedc(handle,dc);
end;
```

OnUnderAfter — Event

This event is triggered after the underlay specified by the user has been painted.

VB Example

```
Private Sub MapPro1_UnderAfter(ByVal dc As Long)
Dim hp As Long
' Get the dc to the PmParent
  h = dc ' GetDC(MapPro1.PmParent)
' Use Windows API MoveToEX
  hp = SelectObject(h, GetStockObject(0))
  MoveToEx h, 200, 1, 0
' Draw Line using Windows APIs from previous point to 100,100
  LineTo h, 1, 200
' Release the dc
  ' ReleaseDC MapPro1.PmParent, h
  SelectObject h, hp
End Sub
```

OnUnderBefore — Event

This event is triggered before the underlay specified by the user has been painted.

OnStatus — Event

This event is triggered during drawing operations and contains a value from 0 to 100 representing the progress of drawing operations. This can be used to show a dynamic bar or some other indicator during lengthy drawing operations.

VB Example

```
Private Sub MAPPRO1_Status(ByVal s As Long)
  ProgressBar1.Visible = True
  ProgressBar1.Value = s
  ProgressBar1.Visible = False
End Sub
```

OpenOverlay(s:String) — Procedure

Loads the specified overlay. Overlay files may be created by the user's application (see Appendixes A and K for format description), or created by Precision Mapping Streets V4.0, and can NOT contain any Metafile Objects. The filetype is determined by the specified filename extension (.ovr or .cmx). By specifying a null name, or a non-existent overlay filename, the control now erases the currently loaded overlay data from memory. For a description of the Overlay file structure, see Appendix "A". The argument needs to be a full file specification, i.e., with drive, path, etc.

VB Example

```
Private Sub Command39_Click()
  '-----------------------------------------------}
  'Display an overlay file created with Precision }
  'Mapping 4.0 (S. Dakota area)                   }
  '-----------------------------------------------}
   MapPro1.OpenOverlay("c:\pmap40\sdak01.ovr")
   ' Zoom to the Overlay area}
   MapPro1.ZoomOverlay
End Sub
```

Delphi

```
Procedure TForm1.Button32Click(Sender: TObject);
{-----------------------------------------------}
{Display an overlay file created with Precision }
{Mapping 4.0 (S. Dakota area)                   }
{-----------------------------------------------}
begin
  MapPro1.OpenOverlay('c:\pmap40\sdak01.ovr');
  {Zoom to the Overlay area}
  MapPro1.ZoomOverlay;
end;
```

OptiRouter:Interface — Interface

Allows the user to acces the OptiRouter Interface (See sections later in this document)

OptiRouterBtn:Boolean — Property

This property controls the visibility of the OptiRouter dialog button in the stock toolbar.

OptiPathOnTop:Boolean — Property

Controls whether the highlighted path painted by the Optirouter is painted on top of all other map layers – if true, (which may obscure place names, highway shields, etc.), or painted before (below) the other map layers.

Overlay:Boolean — Property

Controls visibility of the overlay file, if present. True state sets the overlay to visible.

VB Example

```
Private Sub Form_Load()
  MapPro1.Overlay = True
End Sub
```

Delphi

```
Procedure TForm1.FormCreate(Sender: TObject);
begin
  MapPro1.OverLay:=True;
end;
```

Path_Data1 — Property

Identifies the location of the Precision Mapping Streets coarse data.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Path_data1="d:\pmap40\data1"
End Sub
```

Delphi

```
Procedure TForm1.FormCreate(Sender: TObject);
begin
  MapPro1.Path_data1:='d:\pmap40\data1';
end;
```

Path_Data2 — Property

Identifies the location of the Precision Mapping Streets landmark data.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Path_data2="d:\pmap40\data2"
End Sub
```

Delphi

```
Procedure TForm1.FormCreate(Sender: TObject);
begin
  MapPro1.Path_data2:='d:\pmap40\data2';
end;
```

Path_Data3 — Property

Identifies the location of the Precision Mapping Streets City, ZipCode and Phone database files.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Path_data3="d:\pmap40\data3"
End Sub
```

Delphi

```
Procedure TForm1.FormCreate(Sender: TObject);
begin
  MapPro1.Path_data3:='d:\pmap40\data3';
end;
```

Path_Data4 — Property

Identifies the location of the Points of Interest (POI) database files. Also, note thet the POI0A.SIG file needs to be present in this location for the POI data to be accessible and visible on the map.

Delphi Example

```
procedure TForm1.Button81Click(Sender: TObject);
begin
  Mappro1.path_data4:='c:\mapdata\pmtana0601\data4';
end;
```

Path_Library — Property

Identifies the location of the metafile symbols that are accessed through the CAD toolbar.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Path_library="d:\pmap\pmap.lib"
End Sub
```

Path_States0:String — Property

Identifies the location of States Street data files. The Path_States0 is checked first, then the Path_States1 data directory is checked. If the system finds the data file in the States0 directory it uses that file. If it doesn't find it in the States0, it moves on the the States1. This allows the user to have some data on the hard drive and some on CD. It is also useful when updates are made to certain state files.

Also note that in version 6.0 and above a "XXXXXX.SIG" file must be present in the States data directory based upon the use of either Tiger or TeleAtlas data.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Path_States0="d:\pmap40\states"
End Sub
```

Delphi

```
Procedure TForm1.FormCreate(Sender: TObject);
begin
  MapPro1.Path_states:='d:\pmap40\states';
end;
```

Path_States1:String — Property

Identifies the location of States Street data files. The Path_States0 is checked first, then the Path_States1 data directory is checked. If the system finds the data file in the States0 directory it uses that file. If it doesn't find it in the States0, it moves on to look in the States1 path. This allows the user to have some data on the hard drive and some on CD. It is also useful when updates are made to certain state files.

Also note that in version 6.0 and above a "XXXX.SIG" file must be present in the States data directory based upon the use of either Tiger or TeleAtlas data.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Path_States1="d:\pmap40\states"
End Sub
```

Delphi

```
Procedure TForm1.FormCreate(Sender: TObject);
begin
  MapPro1.Path_States1:='d:\pmap40\states';
end;
```

PhoneRegInfo — Property

This will allow you to modify the text in the Register Over the Phone dialog. This is usefull when you would like a different phone number so users can call your company direct. If this property is set to "" or Null, it will disable the Register Over the Phone button in the registration dialog.

Note: This property may not be visible in the properties inspector, you can only set this via code in your application, usually the form load event.

VB Example

```
Private Sub Command103_Click()
  'Sets the registration phone number to your selected number
 MapPro1.PhoneRegInfo = "Call us at 800-123-4567"
End Sub
```

PmPalette:Integer — Property

PmPalette is the handle of the OCX Palette. Note that this is a read-only property that can be used to synchronize the palette of the user canvas to that of the OCX. Also see the SetOption method section for controlling the PmPallete.

VB Example

```
Private Sub Form_Load()
  MyPalette = MapPro1.PmPalette
' Set user bitmap palette to that of OCX
  Piture1.Picture.hPal = MyPalette
End Sub
```

Delphi

```
Procedure Tform1.button5click(Sender:Tobject)
  MyPalette := MapPro1.PmPalette;
End
```

PmParent:Integer — Property

PmParent is used to assign the parent form handle to the OCX.

VB Example

```
Private Sub Command114_Click()
  ' Get the dc to the PmParent
  h = GetDC(MapPro1.PmParent)
  ' Use Windows API MoveToEX
  MoveToEx h, 1, 1, 0
  ' Draw Line using Windows APIs from previous point to 100,100
  LineTo h, 100, 100
  ' Release the dc
  ReleaseDC MapPro1.PmParent, h
End Sub
```

Delphi

```
Procedure TForm1.Button3click(Sender:Tobject);
  MapPro1.PmParent: = Form1.hwnd;
End
```

PmScale:Double — Property

This property is the same as Scale internal scaling factor and was implemented to avoid conflicts in Visual Basic which already uses a 'Scale' property.

Note: The maximum value that can be set for this property is 10, which results in maximum magnification.

VB Example

```
Private Sub Form_Load()
  MapPro1.PmScale = 10
End Sub
```

POIMgr:Interface — Property

Allows access to the interface that can be used to manage the Points of Interest (POI) database. See POIMgr interface section in this manual. Note that access to the POI data requires explicitly licensing of the POI dataset from UnderTow Software.

PopUpRoute:Boolean — Property

Permits the user to pop up a routing selection menu by pressing the left or right mouse button while holding down the shift

VB Example

```
Private Sub Form_Load()
  ' Disable the popup menu for routing.
  MapPro1.PopUpRoute = False
End Sub
```

Delphi

```
Procedure TForm1.FormCreate(Sender: TObject);
begin
  // Disable the popup menu for routing.
  MapPro1.PopUpRoute:=false;
end;
```

PostUnderlay:Integer — Property

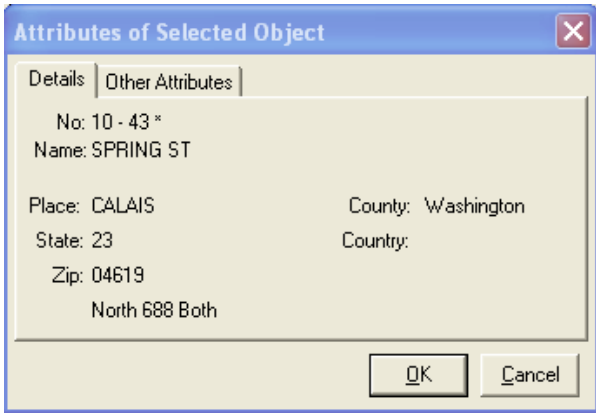
Defines when the Underlay will be painted in the map drawing sequence. The default mode, PostUnderlay=0, operates as in early versions of MapPro, i.e., the Underlay is painted, the water polygons are painted and then all the other data is oainted. When PostUnderlay=1, the order is: the water polygons are painted, then the underlay is painted and then all the other data is painted. When PostInderlay=2, then the all the map data is painted and the Underlay is painted last (before the OnPaintAfter even is fired).

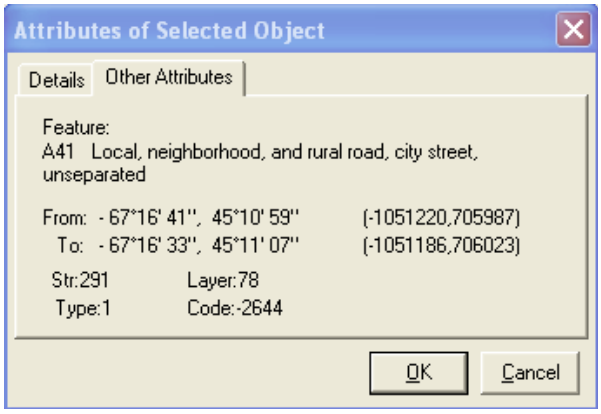
VB Example

```
Private Sub Command71_Click()
 'Open stock dialog to open underlay file
 CommonDialog2.ShowOpen
 s = CommonDialog2.FileName
 ' Set underlay file
 MapPro1.UnderlayFile = s
 ' Set the Underlay transparency mode
 MapPro1.UnderlayTransparent = True
 ' Set the Underlay Transparent color
 MapPro1.UnderlayTrColor = RGB(255, 255, 255)
 'Redraw the Map
 MapPro1.Redraw
End Sub

Private Sub Command70_Click()
' Set the underlay visibility to True
  MapPro1.Underlay = True
  ' Set the Underlay painting Sequence
  MapPro1.PostUnderlay = 2
  MapPro1.Redraw
End Sub
```

QueryObj(x,y:Double) — Property

Searches the current viewport, and if it finds an object within 8 pixels of the specified x.y (Lo,Lat) coordinate, it pops up a dialog with attributes about that object. No action is taken if no object is found within the 8-pixel tolerance.





Redirty — Procedure

Updates the dirty rectangles and the new/changed objects without repainting the complete map surface. Usually called after a user-drawn bitmap is moved. Can provide significant time savings if used following user item movements, rather that using Redraw – which would regenerate the map, or even Refresh, which would repaint all the uset item bitmaps.

VB Example

```
Private Sub Timer2_timer()
  '-----------------------------------------------}
  ' Timer used to show animated car trail        }
```

```
  '-----------------------------------------------}
  dim temp1,temp2 as double
  inc(locnum)
  'Check to see if all cars have been drawn}
  if locnum>MaxCarLoc then
   'Disable the timer to stop the loop}
   timer2.enabled=false
   'Turn AutoQuery back ON}
    MapPro1.AutoQuery=true
   end if
   'Set the bitmap to a new location}
   MapPro1.SetItem(1,carloc[locnum,1],carloc[locnum,2])
   'erase the old bitmap and refresh the one at the new location}
   MapPro1.ReDirty
   'annunciate every other movement}
   if (locnum mod 2) =0 then Messagebeep(0)
End Sub
```

Delphi

```
Procedure TForm1.Timer2Timer(Sender: TObject);
{-----------------------------------------------}
{ Timer used to show animated car trail         }
{-----------------------------------------------}
var temp1,temp2:real;
begin


  inc(locnum);
  {Check to see if all cars have been drawn}
  if locnum>MaxCarLoc then
  begin
    {Disable the timer to stop the loop}
    timer2.enabled:=false;
    {Turn AutoQuery back ON}
    MapPro1.AutoQuery:=true;
   end else
   begin
     {Set the bitmap to a new location}
     MapPro1.SetItem(1,carloc[locnum,1],carloc[locnum,2]);
     {erase the old bitmap and refresh the one at the new
location}
     MapPro1.ReDirty;
   end;
   {annunciate every other movement}
   if (locnum mod 2) =0 then Messagebeep(0);
end;
```

Redraw — Procedure

Causes the map to load any required map data and draw all elements to the specified control. If the map has not changed and only other items, painted on the map, have been modified, using Refresh instead or Redraw would be much faster.

Note: If the users have their own paint handler, then it's better to use the Refresh method, if background drawing is enabled.

VB Example

```
Private Sub Command39_Click()
```

```
  MapPro1.Redraw
End Sub
```

Delphi

```
Procedure TForm1.Update1Click(Sender: TObject);
begin
  MapPro1.redraw;
end;
```

Refresh — Procedure

Causes the map to update its surface with the background bitmap which is saved when background drawing is enabled.

Note: This is different from ReDraw as the screen refresh is done with an image that's already created and doesn't involve the loading of any data.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Refresh
End Sub
```

Delphi

```
Procedure TForm1.SpeedButton6Click(Sender: TObject);
begin
  MapPro1.Refresh;
end;
```

ResizeCtl(dx,dy:Integer) — Procedure

Modifies the internal scale factor and viewing window coordinates to account for changes in the size of the control. Dx and Dy represent the new size that the control adopts after a windows resize operation.

Note: The OCX control may be set to automatically resize to the client window size, in which case the ResizeCtl call might not be required. It is recommended that a call to this procedure occur in response to a Wm_size message which is issued every time a window, control, form, etc. changes size. In a visual development environment, like Delphi, Visual Basic, etc., this message is handled by, or replaced with, an OnResize event.

VB Example

```
Private Sub Command39_Click()
  MapPro1.ResizeCtl(MapPro1.width,MapPro1.height)
End Sub
```

Delphi

```
Procedure TForm1.PmMap1Resize(Sender: TObject);
begin
  MapPro1.ResizeCtl(PmMap1.width,PmMap1.height);
end;
```

Result — Property

This string is used as a general purpose way to return the result of city, areacode and zipcode searches.

Note: A different variable, "Street" is used for Street searches, as they are based on totally different models, i.e., Street searches return multiple hits. Refer to the documentation for each of the search operations to see the prospective contents of "Result".

VB Example

```
Private Sub Command39_Click()
  ' Display the Result from a ZipCode search, in a panel}
  MapPro1.findzip val(text1.text)
  label7.caption=OlPmMap1.Result
End Sub
```

Delphi

```
Procedure TForm2.Button4Click(Sender: TObject);
{ Display the Result from a ZipCode search, in a panel}
var z,code:longint;
begin
  val(edit3.text,z,code);
  MapPro1.findzip(z);
  label7.caption:=MapPro1.Result;
end;
```

RoadOption(Option, Atrib:LongInt) — Procedure

Permits the user to set options to be used for the routing calculation without using the dialog described in the ExecRoadOption method.

The parameters Option and Attrib are described below.

(a) When Option = 1..6, then the value of Attrib sets the travel speed (mph) for the road type identified in Option.

Option	Travel Speed for Type of Road
1	Limited Access Highways (Tiger types A10..A18)
2	US and State Highways (Tiger types A20..A28)
3	Secondary Highways (Tiger types A30..A38)
4	Residential Roads (Tiger types A40..A48)
5	Trails (Tiger types A50..A53)
6	Other (Tiger types A60..A65)

(b) When Option = 17..22, then the value of Attrib sets the priority (0 to 5), of the road type identified in Option,

Option	Priority for Type of Road
17	Limited Access Highways (Tiger types A10..A18)
18	US and State Highways (Tiger types A20..A28)
19	Secondary Highways (Tiger types A30..A38)
20	Residential Roads (Tiger types A40..A48)
21	Trails (Tiger types A50..A53)
22	Other (Tiger types A60..A65)

(c) When Option = 33..38, then the value of Attrib sets the gas mileage of the road type identified in option,

Option	Type of Road
33	Limited Access Highways (Tiger types A10..A18)
34	US and State Highways (Tiger types A20..A28)
35	Secondary Highways (Tiger types A30..A38)
36	Residential Roads (Tiger types A40..A48)
37	Trails (Tiger types A50..A53)
38	Other (Tiger types A60..A65)

(d) When Option = 48, then the value of Attrib sets the color to be used for highlighting the calculated route on the map.

(e) When Option = 49, then the value of Attrib sets the type of hatching to be used for highlighting the calculated route on the map:

0 = Dot (10%)
1 = Dot (20%)
2 = Dot (50%)
3 = Cross
4 = Hatch
5 = Grid

Note: The user should be cautioned that the likelihood of routing failure will increase when road types are totally excluded, i.e., when travel preference priority is set to "0" for one or more road types.

VB Example

```
Private Sub Command39_Click()
  '-----------------------------------------------------}
  ' Change Some Routing options                         }
  '-----------------------------------------------------}
  'change the color of the Routing highlight}
  Hcolor=Hcolor+16
  MapPro1.RoadOption(48,Hcolor)
  'Set faster driving options}
  MapPro1.RoadOption(1,80)
  MapPro1.RoadOption(2,75)
  MapPro1.RoadOption(3,65)
  'Set Road priorities to avoid Limited Access Highways}
  'and preferably use secondary roads}
```

Delphi

```
Procedure TForm1.Button8Click(Sender: TObject);
{-----------------------------------------------------}
{ Change Some Routing options                         }
{-----------------------------------------------------}
begin
  {change the color of the Routing highlight}
  Hcolor:=Hcolor+16;MapPro1.RoadOption(48,Hcolor);
  {Set faster driving options}
  MapPro1.RoadOption(1,80);
  MapPro1.RoadOption(2,75);
```

```
  MapPro1.RoadOption(3,65);
  {Set Road priorities to avoid Limited Access Highways}
  {and preferably use secondary roads}
  MapPro1.RoadOption(17,1);
  MapPro1.RoadOption(18,2);
  MapPro1.RoadOption(19,2);
  MapPro1.RoadOption(20,5);
end;
  MapPro1.RoadOption(17,1)
  MapPro1.RoadOption(18,2)
  MapPro1.RoadOption(19,2)
  MapPro1.RoadOption(20,5)
End Sub
```

Rotate(x:Double) **Procedure**

Rotates the map X degrees about the centerpoint of the view port. Place and Landmarks text remains horizontal. Street label text is automatically rotated so that it aligns itself with the rotated street/road segment. The orientation (from-to) of the string depends on the setting of the Synch boolean property. If Synch is set to false (default), then the text orientation is always restricted in the East quadrants. If Synch is set to true, then the text is rotated through the same angle as the street segment, retaining the original relative orientation to the street. (Also see Synch)

Note: Positive direction is counterclockwise, and zero degrees is due North.

VB Example

```
Private Sub Command39_Click()
  ' Rotate the screen counter clockwise by 20.5 degrees}
  MapPro1.Rotate(20.5)
End Sub
```

Delphi

```
Procedure TForm1.PmMap1Rotate(Sender: TObject);
begin
  { Rotate the screen counter clockwise by 20.5 degrees}
  MapPro1.Rotate(20.5);
end;
```

RouteProgressIcon(FineName:string):Boolean **Procedure**

Allows the user to load their own bitmam to be used in the Routing progress bar (Routing and OptiRouting dialogs), instead of the default, built-in sports car bitmap. It returns true, if the specified bitmap file (24-bit, Windows .BMP) is found, and false otherwise.

RoutingActive:boolean **Property**

Setting the value of this property enables or disables the routing capabilities of the program. Care

should be taking in setting this property because it is tied to the licensing prices with routing, and to the client license registration process. **Please, refer to Apendix N and Appendix O, for more details and client license pricing when the routing capabilities are enabled.**

Delphi Example
```
procedure TForm1.Button75Click(Sender: TObject);
// Toggle Routing capabilities On/Off
begin
  mappro1.RoutingActive:=Not(mappro1.RoutingActive);
  // Note that the routing icon on the toolbar is also toggled
  if mappro1.RoutingActive=false
    then button75.caption:='Routing: Off'
       else button75.caption:='Routing: ON';
end;
```

SaveConfig — Procedure

Saves the current parameters in the configuration file specified in 'Path'.

VB Example
```
Private Sub Command39_Click()
  ' Save config file to d:\MyData
  MapPro1.SaveConfig("d:\MyData\mappro.ocx")
End Sub
```

Delphi
```
Procedure TForm1.SaveBitmap1Click(Sender: TObject);
begin
  // Save config file to d:\MyData
  MapPro1.SaveConfig('d:\MyData\mappro.ocx');
end;
```

SaveExclusion(s:String) — Procedure

Saves the current list of bounding box coordinates to the specified file. This is a flat ASCII file with one entry line per exclusion polygon of the form:

X1, Y1, X2, Y2, Identifying String

The identifying string specified by the user is automatically enclosed in double quotes.

Sample File

-71.068928,42.349952,-71.067840,42.348864,"Route Exclusion-1"
-71.070928,42.351952,-71.069840,42.350864,"Route Exclusion-2"

Delphi
```
Procedure TForm1.Button36Click(Sender: TObject);
// Save currently defined exclusions to file
// Assumed extension: .exl
begin
  MapPro1.SaveExclusion('MyRtExcFile');
end;
```

SaveStreets(s:String) Procedure

Saves the current user defined streets in memory to the specified file.

VB Example

```
Private Sub Command2_Click()
    MapPro1.SaveStreets ("C:\Steets.Str")
End Sub
```

SavetoBitmap(s:String) Procedure

Saves the current background bitmap (the map bitmap) to a BMP file specified by the user.

Note: No extension is added, so the user would have to supply the ".BMP" file extension.

VB Example

```
Private Sub Command39_Click()
  Clipboard.Clear
  MapPro1.SaveToBitmap (App.Path & "\ClipboardSave.bmp")
End Sub
```

Delphi

```
Procedure TForm1.SaveBitmap1Click(Sender: TObject);
begin
  with opendialog1 do
  begin
    filter:='Bitmap Files|*.bmp';
    if execute then
    begin
      MapPro1.SaveToBitmap(filename);
    end;
  end;
end;
```

SavetoGif(s:String) Procedure

Saves the current background bitmap (the map bitmap) to a GIF file specified by the user.

Note: No extension is added, so the user would have to supply the ".GIF" file extension.

VB Example

```
Private Sub Command39_Click()
  MapPro1.SavetoGif (App.Path & "\Test.Gif")
End Sub
```

Delphi

```
Procedure TForm1.SaveBitmap1Click(Sender: TObject);
begin
  with opendialog1 do
  begin
    filter:='Gif Files|*.Gif';
    if execute then
    begin
      MapPro1.SavetoGif(filename);
    end;
  end;
end;
```

SaveViaFile(s:String) Property

Save the currently defined Via points to a file for later use.

VB Example

```
Private Sub Form_Load()
  MapPro1.SaveViaFile "c:\mydata\sample.via"
End Sub
```

SaveView:Boolean Property

Upon starting the application, if true, this property restores the view of the map that was visible when the application was last executed (the coordinates are taken from the .CFG file).

VB Example

```
Private Sub Form_Load()
  MapPro1.SaveView = True
End Sub
```

Delphi Example

```
Procedure TForm1.FormCreate(Sender: TObject);
begin
  MapPro1.SaveView:=True;
end;
```

ScaleProperty:Double

This property is an internal scaling factor and ONLY makes sense when used in a "relative scale" manner. Its value changes when the Magnitude or Miles properties are set. When the scale value is set, the control determines the closest enumerated value of Magnitude. A redraw is issued when the value of this internal scale factor changes.

Note: Note that the maximum value that can be set for this property is 10, which results in maximum magnification. Cannot Use in Visual Basic - Use PMScale.

VB Example

```
Cannot Use in Visual Basic - Use PMScale
```

Delphi Example

```
Procedure TForm1.Button8Click(Sender: TObject);
begin
  // echo current value, and reduce scale by 50%
  label1.caption:=floattostr(mappro1.scale);
  mappro1.scale:=mappro1.scale*0.5;
  Mappro1.Redraw;
  // echo new scale value
 label2.caption:=floattostr(mappro1.scale);
end;
```

ScaleBar:Integer — Property

An easier way to display the scale bar without the need to use the DC and a specific offset location. The available enumerated values for this option are:

0 = sbNone
1 = sbTopLeft
2 = sbTopRight
3 = sbBottomLeft
4 = sbBottomRight

Screen_Aspect:Double — Property

Defines the aspect ratio used when drawing to the OCX control. This is the ratio of Width to Height.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Screen_Aspect=2
End Sub
```

Delphi

```
Procedure TForm1.Button1Click(Sender: TObject);
begin
  MapPro1.Screen_Aspect:=2;
end;
```

SetDirtyRect(LeftX,TopY,RightX,BottomY) — Procedure

Permits users to add their own rectangle to the dirty rectange list which can then be updated with the ReDirty method.

VB Example

```
Private Sub Command6_Click()
  ' Set a small rectangle of current screen to be preserved
  MapPro1.SetDirtyRect 2, 2, 100, 100
  ' Get the dc and paint the main screen
  dc = GetDC(MapPro1.Handle)
  hp = SelectObject(dc, GetStockObject(0))
  Rectangle dc, 1, 1, 700, 500
  SelectObject dc, hp
  ' Issue the redirty command.  The added rectangle should
  ' be retained, while the rest of the screen is white
  MapPro1.Redirty
End Sub
```

Delphi

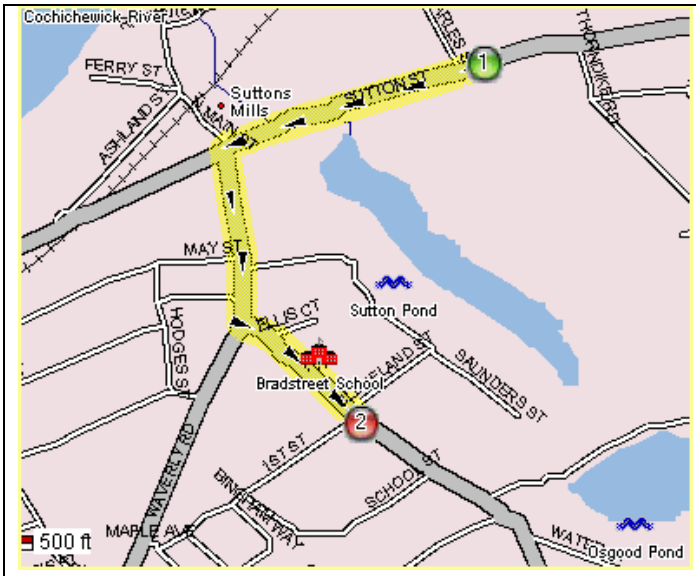
```
procedure TForm1.Button6Click(Sender: TObject);
var hp:hbrush;
  dc:integer;
```

```
begin
  // Set a small rectangle of current screen to be preserved
  pmap21.SetDirtyRect(2,2,100,100);
  // Get the dc and paint the main screen black
  dc:=getdc(Pmap21.handle);
  hp:=selectobject(dc,getstockobject(black_brush));
  windows.rectangle(dc,1,1,700,500);
  selectobject(dc,hp);
  // Issue the redirty command.  The added rectangle should
  // be retained, while the rest of the screen is black.
  pmap21.Redirty;
end;
```

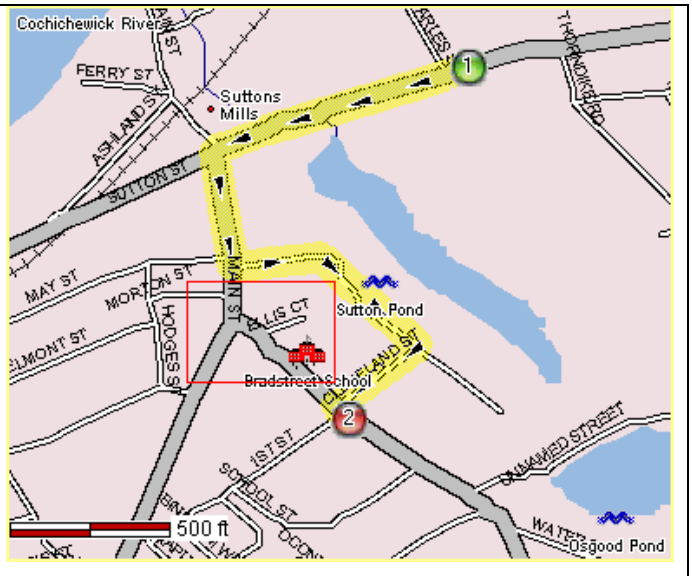
SetExclusion(IDString:string; x1,y1,x2,y2:double) **Procedure**

Adds the bounding box defined by the user-specified string and the two corners (x1,y1 and x2,y2) to the memory-based routing exclusion list. (units are in degrees). Note that any such exclusions are lost when the application is terminated, unless they have been previously saved to a file using the SaveExclusion method.

The X1,Y1 and X2,Y2 coordinate pairs specify a bounding polygon for the exclusion. Road segments with both end-nodes within the exclusion polygon, are excluded from the routing nodes list.



Original Calculated Route



Calculated Route after Exclusion (red rectangle) is set

Delphi Example

```
procedure TForm1.Button32Click(Sender: TObject);
// Add two exclusion polygons to the current exclusions list
var id:string;
begin
 ExcludeRt:=true;
 exx1:=strtofloat(edit6.text);
 eyy1:=strtofloat(edit7.text);
 exx2:=strtofloat(edit13.text);
```

```
   eyy2:=strtofloat(edit14.text);
   id:=edit15.text;
   MapPro1.SetExclusion(id+'-1', exx1, eyy1, exx2, eyy2);
   // Set another exclusion slightly offset from first one
   MapPro1.SetExclusion(id+'-2', exx1-0.002, eyy1+0.002,
                        exx2-0.002, eyy2+0.002);
   MapPro1.ZoomWindow(exx1-0.002,eyy1+0.002,exx2-0.002,eyy2+0.002);
   panel6.visible:=false;
  end;
```

SetItem(id: Integer; x, y: OleVariant) **Procedure**

Creates an overlay object with the specified ID at the lon/lat coordinates. If an item of the same ID already exists, the item is repositioned to the new coordinates. These "items", sometimes also refered to as "user items", can be used in a variety of application, e.g., displaying movement of an automobile on a map, as shown in the VB example below.

Note: The SetItem procedure needs to be executed PRIOR to using the SeItemLocalBitmap procedures. The maximum number of elements is 32,000, but the maximum value of 'id' has to be less than a word.

VB Example

```
Private Sub Command110_Click()
  'Cycle through the various mapmodes, when mapmode is 1, then
  ' click on the map to record positions to temporary array
  MapPro1.MapMode = MapPro1.MapMode + 1
  If MapPro1.MapMode > 4 Then MapPro1.MapMode = 0
  Command110.Caption = "Mode=" & MapPro1.MapMode
End Sub


Private Sub MapPro1_Click()
  ' If the mapmodes is MdUser, then add the coordinates
  ' of the point that is clicked on, to a temporary
  ' two-dimentional array (needs to have been predefined)
  If MapPro1.MapMode = MdUser Then
    ipos = ipos + 1
    pos(ipos, 1) = MapPro1.Xcord
    pos(ipos, 2) = MapPro1.Ycord
  Else
  End If
End Sub


Private Sub Command111_Click()
  ' Set the starting index
  Iposstart = 0
  ' Set the timer interval
  Timer1.Interval = 200
  ' ebnable the timer
  Timer1.Enabled = True
End Sub

Private Sub Timer1_Timer()
' Check to see if all the points have been plotted
If Iposstart > ipos Then
  Timer1.Enabled = False
```

```
Else
  'If Not, set the item to the next set of coordinates
  Iposstart = Iposstart + 1
  MapPro1.SetItem 101, pos(Iposstart, 1), pos(Iposstart, 2)
  MapPro1.SetItemLocalBitmap 101, "d:\test\mappro71\CarIcon.bmp"
  MapPro1.SetItemString 101, Iposstart
  MapPro1.Refresh
End If
End Sub
```

Delphi

```
Procedure TForm1.Button7Click(Sender: TObject);
begin
  MapPro1.SetItem(1,-120,32);
  MapPro1.SetItemLocalBitmap(1,'BMP1.BMP');
end;
```

SetItem2Back(id:Integer) — Procedure

Sets the user-created object with "Id" to be painted on the screen first when a screen update takes place.

VB Example

```
Private Sub Command39_Click()
  MapPro1.SetItem2Back(12)
End Sub
```

Delphi

```
Procedure TForm1.Button4Click(Sender: TObject);
begin
  MapPro1.SetItem2Back(12);
end;
```

SetItem2Front(id:Integer) — Procedure

Sets the user-created object with "Id" to be painted on the screen last (on top) when a screen update takes place.

VB Example

```
Private Sub Command39_Click()
  MapPro1.SetItem2Front(12)
End Sub
```

Delphi

```
Procedure TForm1.Button4Click(Sender: TObject);
begin
  MapPro1.SetItem2Front(12);
end;
```

SetItemAngle(id:integer;x:Double) — Procedure

Allows the user to rotate (clockwise) a user bitmap placed on the screen using SetItem.

Note: The color of the lower left pixel of the bitmap becomes transparent when rotated. It should be further noted that any bitmap that is not masked (early MAPOCX implementations required it to be masked for transparency), sets the color of the lower left pixel to transparent when placed on the screen using SetItem.

VB Example

```
Private Sub Command1_Click()
  xc = (MapPro1.LonRight + MapPro1.LonLeft) / 2
  yc = (MapPro1.LatTop + MapPro1.LatBottom) / 2
  ' Set the item bitmap
  MapPro1.SetItem 1,xc,yc
  MapPro1.SetItemLocalBitmap 1,"\images\sample.bmp"
  ' Rotate the bitmap 45 degrees
  MapPro1.SetItemAngle 1,45
  ' Use Redirty instead of having to repaint the whole screen
  MapPro1.Redirty
End Sub
```

SetItemBitmap(id:integer; handle:integer) — Procedure

Attaches a user bitmap by assigning the handle (id#) of the bitmap (h) to the object. A mask concept is used for bitmaps. If SetItemBitmap or SetItemLocalBitmap are passed a "negative" id #, they do NOT mask the bitmap, they simply paint the square bitmap on the screen. All the user has to do is to simply negate the item #. If the ID is positive, it will mask the color that is used in the first pixel (top left). This allows for a transparant background.

Note: The SetItem procedure needs to be executed prior to using the SetItemBitmap procedures. The user is responsible for allocating and deallocating the bitmap resource. It should also be noted that on systems with 256 colors or less, 16 color bitmaps should be used (see Appendix "B"), otherwise the bitmap palette might interact with the background or the OCX control palette.

There is a limit of 200 unique bitmaps that can be used by the item caching system. Attempts to use more than that, may result in the User Items layer not rendering.

VB Example

```
Private Sub Command39_Click()
  MapPro1.SetItem 1,-100,32
  'Set bitmap from existing handle
  MapPro1.Setitembitmap(1,Image1.Picture.handle)e
End Sub
```

Delphi

```
Procedure TForm1.Button7Click(Sender: TObject);
begin
  MapPro1.SetItem(1,-100,32);
  {Set bitmap from existing handle}
  MapPro1.Setitembitmap(1,Image1.Picture.Bitmap.handle);
  {Draw on the bitmap surface}
  with Image1.Picture.bitmap.canvas do
  begin
    pen.color:=clred;
    Moveto(0,0);
    Lineto(20,20);
  end;
end;
```

SetItemLocalBitmap(id:integer; s:String) — Procedure

Causes the specified bitmap file (s) to be loaded into memory and attaches the bitmap handle to the overlay object. A mask concept is used for bitmaps. If SetItemBitmap or SetItemLocalBitmap are passed a "negative" id #, they do NOT mask the bitmap, they simply paint the square bitmap on the screen. All the user has to do is to simply negate the item #. If the ID is positive, it will mask the color that is used in the first pixel (top left). This allows for a transparant background.

Note: The SetItem procedure needs to be used to set an item ID before this procedure is called to assign the bitmap to it. The number of unique local bitmaps that may be set is 200. Using more than that may result in the User Item layer not rendering.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Gotopoint -120,32
  MapPro1.SetItem 1,-120,32
  call MapPro1.SetitemLocalbitmap (1,"bmp1.bmp")
End Sub
```

Delphi

```
Procedure TForm1.Button7Click(Sender: TObject);
begin
  MapPro1.Gotopoint(-120,32);
  MapPro1.SetItem(1,-120,32);
  MapPro1.SetitemLocalbitmap(1,'bmp1.bmp');
end;
```

SetItemString(id:integer; s:String) — Procedure

Attaches a descriptive string to the overlay object created by the user. Using a CR will act as a line break. If no bitmap is selected, rather than getting a black frame, only the text assigned to the item is displayed, centered, and enclosed in a black

VB Example

```
Private Sub Command39_Click()
  Call MapPro1.SetItem (1,-120,32)
  Call MapPro1.SetItemString(count,"Item:" + str(count) & chr(13)
& "Break")
End Sub
```

Delphi

```
Procedure TForm1.Button7Click(Sender: TObject);
begin
  MapPro1.SetItem(1,-120,32);
  MapPro1.SetItemString(count,'Item:'+inttostr(count));
end;
```

SetItemVis — Procedure

Sets the attribute for the specified user-created object to visible (when flag is true) or invisible (when Flag is false).

VB Example

```
Private Sub Command39_Click()
  '-----------------------------------------------}
  'Set the Visibility of the yellow cars to True  }
  '-----------------------------------------------}
  dim locnum as integer
  for LocNum = 9 to MaxCarLoc
  If caronscreen = true then
    if (Locnum mod 2)=0 then
     MapPro1.SetitemVis(LocNum,true)
    else
    MapPro1.SetitemVis(LocNum,false)
    end if
   else
   'If the cars have not been drawn, display a message}
   Call MsgBox("No cars drawn on the map", vbCritical, "No Cars”
   MapPro1.refresh
 Exit Sub
```

SetOption(OpCode, Option:LongInt):LongInt — Function

Allows the user to change a number of OCX properties as identified by the OpCode and Option values given below (in hex notation). It returns a Long Integer which contains useful information depending on the OpCode.

Helpful Hint (for VB Users)

Users have reported problems with setting these OpCodes in VB. For example, when specifying an OpCode of EEEE, the **&HEEEE** Vb representation, by basic positional values math, is supposed to be

$(14x16^3)+(14x16^2)+(14x16^1)+(14x16^0) = 61166$

However, in VB , this evaluates to -4270, because, for some reason, VB Does not appear to use Unsigned values for Hex operations. Users may implement their own workarounds, but here is a VB function that can help.

```
Public Function ConvertSigned2Unsigned(N As Long) As Double
 If N < 0 Then
    If (N < &H8FFF) Then
        ConvertSigned2Unsigned = (2 ^ 32) + N
    Else
        ConvertSigned2Unsigned = (2 ^ 16) + N
    End If
 Else
   ConvertSigned2Unsigned = N
 End If
End Function
```

If you call *SetOption(ConvertSigned2Unsigned(&HEEEE),1)* you should get the desired results.

Note that you can use this function for any SetOption and any other situation where VB gets confused in trying to handle unsigned values. Conversely, you can calculate the hex values out and call the SetOption routines with decimal values in VB, e.g., SetOption 61166,2

OpCode = 00 (Special Color Options)

Special Opcode use to reset or query attributes according to the "Option" values shown below. (Note that only some of the OpCodes are shown here, but specifying OpCode 00, the user can query the value of *any* OpCode described in this section, by setting the appropriate Option value).

Option = -1
Reset the palette to the built-in default. The function returns zero if the operation was successful.

Option = 00
Regenerate the palette handle. Required since the palette is not automatically generated every time a
color is added. The function returns zero if the operation was successful.

Option = 01
Returns the value of the background color. For example, BkCol=.Setoption($00,01) would return an integer representing the RGB color of the background.

Option = 02
Returns the value of the water color.

Option = 03
Returns the value of the parks color.

Option = 10..1F
Returns the color value of the shade color specified (used for shading states and counties).

Option = 20..6F
Returns the color of the specified state, (see below for State ID number). Note that either a color or a color index will be returned. If a 3-byte value is returned, then it is an RGB color. If a 4-byte value is returned with the highest byte equal to $08, then the number is an index to the built-in shading color definition.

Option = 80..84
Returns the value of the corresponding street labeling option (see OpCodes 80..84 below for a detailed
description).

Option = $89
Returns the value corresponding to the current visibility state of the PMAP built-in coarse layer (see OpCode $89 below for description).

Option = $100+N
Returns the value corresponding to the current visibility state of layer N (see OpCode $100+N below for layer description).

Option = $EEE2

Returns an integer indicating the current graphics bitmap mode (see the description for OpCode $EEE2 later in this section for a description).

OpCode = 01 (Background Color)
Change the background color (Canada, Mexico, etc.), to that specified by Option. (The default is yellow), e.g., .SetOption($01,$00FF00) would set the color to green.

OpCode = 02 (Water Color)
Change the color of the water (ocean) to that specified by Option (The default is blue) , e.g., .SetOption($01,$00FF00) would set the color to green.

OpCode = 03 (Parks Color)
Change the color of the park areas to that specified by Option (The default is green) , e.g., .SetOption($01,$00FF00) would set the color to green.

OpCode = 10...1F (Build-in Color Shades)
Set the identified color, used for shading states and county, areas to that specified by Option. Up to 16 dithered colors are used for the shading. For example, .SetOption($10,$00FF00) would set the first color in the shading array to green. Use SetOption(0,1) if you need to reset the default palette.

OpCode = 20...5F (State Colors)
Set the color for the state specified by the OpCode (see table below), to that specified by Option. Option can be an RGB color or it can be an index to the shading colors, if a 4-byte value is specified and the highest byte is $08. For example, using .SetOption($20,$FF0000) (in Delphi) would set the color of Alabama to blue, whereas .SetOption($46,$08000005) would set the color of Ohio to the 5-th value in the color shading array (see opcodes $10 - $1F above).

State/#	State/#	State/#	State/#	*Province/#*
AL 20	KS 33	OH 46	OH 46	*AB 5B*
AK 21	KY 34	OK 47	OK 47	*BC 5C*
RSV 22	LA 35	OR 48	OR 48	*MB 48*
AZ 23	ME 36	PA 49	PA 49	*NB 5E*
AR 24	MD 37	RSV 4A	RSV 4A	*NF 5F*
CA 25	MA 38	RI 4B	RI 4B	*NS 60*
RSV* 26	MI 39	SC 4C	SC 4C	*ON 61*
CO 27	MN 3A	SD 4D	SD 4D	*PE 62*
CT 28	MS 3B	TN 4E	TN 4E	*QC 63*
DE 29	MO 3C	TX 4F	TX 4F	*YT 64*
DC 2A	MT 3D	UT 50	UT 50	*NT 65*
FL 2B	NE 3E	VT 51	VT 51	*SK 66*
GA 2C	NV 3F	VA 52	VA 52	
RSV 2D	NH 40	RSV 53	RSV 53	
RSV 2E	NJ 41	WA 54	WA 54	
ID 2F	NM 42	WV 55	WV 55	
IL 30	NY 43	WI 56	WI 56	
IN 31	NC 44	WY 57	WY 57	
IA 32	ND 45			

* RSV Denotes slot is reserved for later use

OpCode = $80 (Major Street Labeling)
Set the labeling of major streets (secondary roads and above) to start when the scale reaches the value specified by Option (in tenths of miles). For example, SetOption($80,5) would set the major street labeling to start when the scale is 5/10=0.5 miles. Of course, such roads are labeled only if they are visible at the specified scale.

OpCode = $81 (Minor Street Labeling)
Set the labeling of minor (neighborhood) street to start when the scale reaches the value specified by Option (in tenths of miles). For example, SetOption($81,2) would set the minor street labeling to start when the scale was 2/10=0.2 miles.

OpCode = $82 (Global Street Labeling)
Set the labeling of roads (major and minor) ON, when the specified Option value is 1, or OFF, when the specified Option value is 0. For example, SetOption($82,0) would set street labeling OFF.

OpCode = $83 (Street Label Interference)
Set the simple, built-in label interference detection for street labeling of roads (major and minor) ON, when the specified Option value is 1, or OFF, when the specified Option value is 0. For example, SetOption($83,1) would set street labeling interference detection ON.

OpCode = $84 (Hide Suffix)
Hide Suffix for street labeling of roads (major and minor) is ON, when the specified Option value is 1, or OFF, when the specified Option value is 0. For example, SetOption($84,0) would set street labeling Hide Suffix OFF. With this option set ON, a road segment normally labeled as “Melton Ave” would now be lebeled simply “Melton”.

OpCode = $85 (Street Text Size)
Scale Factor for the street labeling text. These values are (mod 5) so a value of 10 increases the size of the font used for labeling streets by a factor of 2.0.

OpCode = $86 (User Item Visibility)
Turns the visibility of all user items, created by the SetItem methods, Off (Option=0), or On (Option=1).

OpCode = $88 (Display Highway Shields)
Controls the display of Highway Shields. Shield display is ON, when the specified Option value is 1, or OFF, when the specified Option value is 0.

OpCode = $89 (Display Coasre Layer)
Turn the complete PMAP built-in coarse layer ON/OFF. SetOption($89,1) - Sets PMAP coarse layer visible. SetOption($89,0) - Sets PMAP coarse layer invisible

OpCode = $8F (Display Bounding Polygons)
Display the simple bounding polygons that are used for placing text on the map and performing a simple collision detection. Primarily used for debugging purposes. The bounding polygon is displayed when the specified Option value is 1, or OFF, when the specified Option value is 0.

OpCode = $90 (State Text Size)
Sets the size of the font used to label the 2-letter State name abbreviations and the major cities in each state. The default size is 11.

OpCode = $91 (Place Name Text Size)
Sets the size of the font used to label place names. The default size is 10.

OpCode = $92 (Landmark Text size)
Sets the size of the font used to label Landmarks when zoomed below 2 miles. The default size is 9.

OpCode = $93 (Reserved)
Reserved for later use.

OpCode = $94 (Highway Shield Text Size)
Sets the size of the font used to label Highway Shields. The default size is 12. The shield is resized to accommodate the new specified text size.

OpCode = $95 (Street Name Text Size)
Sets the text size for the Street names. The default value is 9.

OpCode = $98 (Street Data Threshold)
Sets the scale value (miles) at which the coarse data layers become invisible and the Street level data layers phase in. The default values is 2.0 miles. The maximum value that this is internally limited to, is 4 miles.

OpCode = $99 (City Label Spacing)
Sets the spacing between City labels (in points). The larger this value is, the more space will be reserved between city name labels during the simple collision detection algorithm of the control.

OpCode = $100+N (Layer Visibility)
Set the visibility of specific layers ON/OFF. All layer values have an offset of $100, and are listed as the increment above $100. The form is SetOption($100+N,State) where "N" is one of the values below, and "State" is the visibility of the layer, 0 (false), or 1 (true).

For example, the Delphi call: MapPro1.SetOption($100+33,byte(false))
will set the display of highway shields OFF.

N = 1 State Boundaries	N = 21 Reserved
N = 2 State Interiors (shading)	N = 22 Reserved
N = 3 Reserved	N = 23 Permanent Hydro Features
N = 4 Reserved	N = 24 Intermitent Hydro Features
N = 5 County Boundaries	N = 25 Reserved
N = 6 MCD Boundaries	N = 26 Reserved
N = 7 Place Boundaries	N = 27 Reserved
N = 8 MCD Shading	N = 28 Reserved
N = 9 Interstate Highways	N = 29 Reserved
N = 10 US Principal Highways	N = 30 Reserved
N = 11 Secondary (County) Highways	N = 31 Reserved
N = 12 Secondary (State) Highways	N = 32 Reserved
N = 13 Other Main Roads	N = 33 Shields (Highways, State/County)
N = 14 Streets	N = 34 Coarse Secondary Highways
N = 15 Landmarks	N = 35 Reserved
N = 16 Reserved	N = 36 Reserved
N = 17 Railroads	N = 37 Reserved
N = 18 Power Lines	N = 38 Reserved
N = 19 Parks	N = 39 Reserved
N = 20 Reserved	N = 40 Reserved

SetOption($100+N), where N=41 and above, controls individual layers from the Tiger Data Set, as follows (see Appendix for Census Feature Class Code [CFCC]descriptions):

N/CFCC	N/CFCC	N/CFCC	N/CFCC	N/CFCC	N/CFCC	N/CFCC	N/CFCC
41 A00	65 A26	89 A53	113 B31	137 D30	161 D65	185 F20	209 H13
42 A01	66 A27	90 A60	114 B32	138 D31	162 D66	186 F21	210 H20
43 A02	67 A28	91 A61	115 B33	139 D32	163 D70	187 F22	211 H21
44 A03	68 A30	92 A62	116 B40	140 D33	164 D71	188 F23	212 H22
45 A04	69 A31	93 A63	117 B50	141 D34	165 D80	189 F24	213 H30
46 A05	70 A32	94 A64	118 B51	142 D35	166 D81	190 F25	214 H31
47 A06	71 A33	95 A65	119 B52	143 D36	167 D82	191 F30	215 H32
48 A07	72 A34	96 A70	120 C00	144 D37	168 D83	192 F40	216 H40
49 A08	73 A35	97 A71	121 C10	145 D40	169 D84	193 F50	217 H41
50 A10	74 A36	98 A72	122 C20	146 D41	170 D85	194 F60	218 H42
51 A11	75 A37	99 A73	123 C30	147 D42	171 D90	195 F70	219 H50
52 A12	76 A38	100 B00	124 C31	148 D43	172 D91	196 F71	220 H51
53 A13	77 A40	101 B01	125 D00	149 D44	173 E00	197 F72	221 H53
54 A14	78 A41	102 B02	126 D10	150 D50	174 E10	198 F73	222 H60
55 A15	79 A42	103 B03	127 D20	151 D51	175 E20	199 F74	223 H70
56 A16	80 A43	104 B10	128 D21	152 D52	176 E21	200 F80	224 H71
57 A17	81 A44	105 B11	129 D22	153 D53	177 E22	201 F81	225 H72
58 A18	82 A45	106 B12	130 D23	154 D54	178 F00	202 F82	226 H73
59 A20	83 A46	107 B13	131 D24	155 D55	179 F10	203 H00	227 H74
60 A21	84 A47	108 B20	132 D25	156 D60	180 F11	204 H01	228 H75
61 A22	85 A48	109 B21	133 D26	157 D61	181 F12	205 H02	229 H80
62 A23	86 A50	110 B22	134 D27	158 D62	182 F13	206 H10	230 H81
63 A24	87 A51	111 B23	135 D28	159 D63	183 F14	207 H11	231 X00
64 A25	88 A52	112 B30	136 D29	160 D64	184 F15	208 H12	

OpCode = $200 (State Polygon Shading)
Shade State/Province Polygons. Shading is performed using the currently defined color shades (See Appendix for information on the shading array stored in the configuration file)

OpCode = $201 (County Polygon Shading)
Shade County Polygons. Shading is performed using the currently defined color shades (See Appendix for information on the shading array stored in the configuration file)

OpCode = $202 (MCD Polygon Shading)
Shade MCD Polygons. Shading is performed using the currently defined color shades (See Appendix for information on the shading array stored in the configuration file)

OpCode = $203 (Place Polygon Shading)
Shade Place Polygons. Shading is performed using the currently defined color shades (See Appendix for information on the shading array stored in the configuration file)

OpCode = $301 (State Boundary Color)
Sets the pen color to be used for drawing State boundaries.

OpCode = $302 (County Boundary Color)
Sets the pen color to be used for drawing County boundaries.

OpCode = $303 (MCD Boundary Color)
Sets the pen color to be used for drawing MCD boundaries.

OpCode = $C001 (Debug Mode)
It controls the OCX debug mode. If Option=1 the debug mode is turned ON, and if Option=0, it is turned OFF. When the debug mode is ON, a file named **mpdebug.txt** is created in the default directory of the C: drive, containing useful information for Undertow Software's development department, when trying to resolve complex tech support problems. This file can get fairly large. Use this option only when directed to do so by Undertow Software, and follow whatever other instructions are given to you at the time.

OpCode = $C002 (Render Time)
Time (Sec) to generate the last display, including the time to load the data.

OpCode = $C003 (Load Data Time)
Time (sec) to load the data (included in value returned by OpCode $C002).

OpCode = $C004 (World Extents)
Setting it to true, permits the control to accommodate coordinates outside North America.

OpCode = $D001 (Memory Usage)
Maximun Number of EMS (the internal memory management module) Pages Allocated for any operation this far (ReadOnly, can be queried with OpCode 00).

OpCode = $D002 (Memory Page Size)
Size of EMS page in bytes (ReadOnly, can be queried with OpCode 00). Note that the page size and the maximum allowable pages is optimized in each release of MapPro, in order to support the widest group of hardware and also perform efficiently.

OpCode = $D003 (Free Memory)
Free EMS, in bytes (ReadOnly, can be queried with OpCode 00).

OpCode = $D004 (Max Memory)
Maximum EMS, bytes, allocated (ReadOnly, can be queried with OpCode 00).

OpCode = $D005 (One Way Arrow Collision)
This controls the one way street indicator collision bounding rectangle size. This can limit the number of arrows and overlap. You can do so using SetOption($D005,N) where N adds the specified number of pixels to the rectangle bounding box.

OpCode = $D007 (Min Zoom Scale)
Sets the minimum zoom scale value (in feet). There is also a built-in limit of 100 feet.

OpCode = $D008 (Grid File Memory Usage)
Returns the memory (bytes) used by the currently loaded grid files.

OpCode = $EEE2 (Internal Bitmap Mode)
Set the mode to be used by the map drawing module, in terms of the Windows PixelFormats. This allows the user to select the type of internal bitmap to be generated by the map drawing routine. If the developer wants to use 16- or 24-bit bitmaps generated, instead of the default 8-bit 256-color bitmaps, they need only set the appropriate mode.

Option can be used to set one of the following modes:

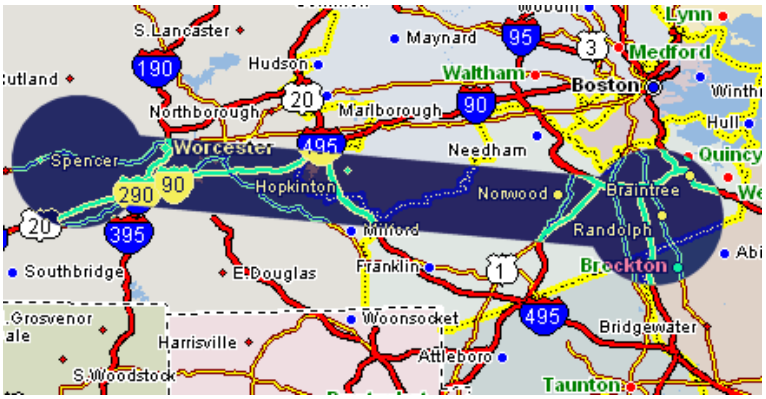
Option = 1, pf1bit - Mono
Option = 2, pf4bit - 16 color
Option = 3, pf8bit - 256 color
Option = 4, pf15bit - 15 bit color
Option = 5, pf16bit - 16 bit color
Option = 6, pf24bit - RGB True color
Option = 7, pf32bit - 32 bit color

OpCode = $EEE4 (User Item Paint Order)
Sets the order in which the OCX paints the user items on screen. Option = 0 - Instructs the OCX to paint the items in the OnPaintAfter event "prior" to painting the user bitmaps placed by the SetItem methods. Option = 1 instructs the OCX to paint the user bitmaps first. The values are in Hex.

OpCode = $EEE9 (Double Lined Roads)
Set the single-line street mode when zoomed very close. The default is Option=0, i.e., double line streets are used. If Option is set to 1, then single line streets are used when zoomed in.

OpCode = $EEEA (Routing Local Radius)
Sets the local radius (through the Option value, in miles) to be used at the start/end points or a specified routing calculation, for connecting to the local road network. The larger the Radius, the

higher the likelihood that the Interstate Network will find a node to connect to the local road network. Experienced user, familiar with the routing capabilities of the program can use this option to optimize their routing calculations, or eliminate problems where the default radius is too small and a national to local transition node is not found. The default value is 10 mi.



When doing routing calculations, a straight line is drawn between the **From** and **To** points. If applicable, an highway route is calculated between the points and then a transition from a highway to a local street node is sought for within the radius set with this OpCode around the **From** and **To** points.

OpCode = $EEEB (Routing Band Width)

Sets the width of the bitmap used to determine the grids to load for en-route routing. The larger the width, the higher the likelihood that a route calculation will complete, at the expense of consuming significantly higher resources. The default value is set to 100 mi. This is the width of the straight line drawn between the **From** and **To** points, and it defines how many grids need to be loaded in creating the node network for calculating the route (See graphic for OpCode $EEEA).

OpCode = $EEEC (Routing Highlight Resolution)

If Option=1, then ALL the nodes are used for highlighting the route, and if Option=0, then only the first and last points of each chain are used, as it used to work in earlier versions of the OCX. The default value is set to 0.

OpCode = $EEEE (FindClosest Options)

It affectes the behavior when using the FindClosest Method. If Option=0, then it operates as before, i.e., it returns the coordinate of the segment endpoint. If Option=1, then it returns the coordinates of the interpolated address (Note that since not all segments are divided the same way in real life, the interpolated coordinates may, or may not be correct).

Some simple code examples using the SetOption method.

VB Example

```
Private Sub Command39_Click()
  '-------------------------------------------------}
  ' Change the background (Canada/Mexico) color    }
  '-------------------------------------------------}
  dim flag as long
  'Set the color using the RGB function}
  flag=MapPro1.setoption(1,RGB(33,23,45))
```

```
  'repaint the map surface to reflect the new color}
   MapPro1.redraw
End Sub


  '-------------------------------------------------}
  ' Set Some options using SetOption                }
  '-------------------------------------------------}
  dim flag as Integer
  dim cl,StNum as Long
  'Set the State number for MA }
  StNum=38
  'Set the color to solid Red}
  cl=RGB(255,0,0)
  'Use SetOption to Set the color of MA to Red}
  flag=MapPro1.SetOption(StNum,cl)
  'Set the Palette}
  MapPro1.SetOption(0,0)
  'Check to see if the operation was successful and echo
  'appropriate message as a caption to panel 5}
   if flag<>0 then Panel5.caption=""Palette Setting Failed!""
   else Panel5.caption=""Palette Setting Successful!""
  'If the operation was successful, redraw the map
  'to reflect the new color}
  if flag=0 then MapPro1.redraw
End Sub


  '-------------------------------------------------}
  ' Example using SetOption to query the map control }
  '-------------------------------------------------}
  Dim val,QResult,opt as long
  'Convert the input entered by user in Textbox #3 to a value}
  Opt = val(Text3.text)
  'Use SetOptions to query the property specified by the user}
  QResult=MapPro1.SetOption(0,Opt)
  'Echo the returned result as the panel5 caption}
  panel5.caption=str(opt)+"":""+str(QResult)
End Sub
```

Delphi Example

```
Procedure TForm1.Button14Click(Sender: TObject);
{-------------------------------------------------}
{ Change the background (Canada/Mexico) color    }
{-------------------------------------------------}
var flag:longint;
begin
  {Set the color using the RGB function}
  flag:=MapPro1.setoption(1,RGB($33,$23,$45));
  {repaint the map surface to reflect the new color}
  MapPro1.redraw;
end;

procedure TForm1.Button15Click(Sender: TObject);
{-------------------------------------------------}
{ Set Some options using SetOption                }
{-------------------------------------------------}
var flag:integer;
    cl,StNum:LongInt;
begin
  {Set the State number for MA }
```

```
  StNum:=$38;
  {Set the color to solid Red}
  cl:=RGB($ff,0,0);
  {Use SetOption to Set the color of MA to Red}
  flag:=MapPro1.SetOption(StNum,cl);
  {Set the Palette}
  MapPro1.SetOption(0,0);
  {Check to see if the operation was successful and echo
   appropriate message as a caption to panel 5}
  if flag<>0 then Panel5.caption:='Palette Setting Failed!'
    else Panel5.caption:='Palette Setting Successful!';
  {If the operation was successful, redraw the map
  to reflect the new color}
  if flag=0 then MapPro1.redraw;
end;

procedure TForm1.Button17Click(Sender: TObject);
{-------------------------------------------------}
{ Example using SetOption to query the map control }
{-------------------------------------------------}
var QResult,opt:longint;
begin
  {Convert the input entered by user in editbox #3 to a value}
  val(edit3.text,Opt,code);
  {Use SetOptions to query the property specified by the user}
  QResult:=MapPro1.SetOption(0,Opt);
  {Echo the returned result as the panel5 caption}
  panel5.caption:=inttostr(opt)+':'+inttostr(QResult);
end;
```

Shade_Cnty:Boolean — Property

If set to True, the polygons defined by the County outlines are shaded. If set to false, they are not shaded, but instead assume the generic background color.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Shade_Cnty=false
End Sub
```

Delphi Example

```
Procedure TForm1.Button11Click(Sender: TObject);
begin
  MapPro1.Shade_Cnty:=true;
end;
```

Shade_MCD:Boolean — Property

If set to True, the polygons defined by the MCD outlines are shaded. If set to false, they are not shaded, but instead assume the generic background color.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Shade_MCD=false
```

```
End Sub
```

Delphi Example

```
Procedure TForm1.Button10Click(Sender: TObject);
begin
  MapPro1.Shade_MCD:=true;
end;
```

Shade_Plc:Boolean — Property

If set to True, the polygons defined by the Place outlines are shaded. If set to false, they are not shaded, but instead assume the generic background color.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Shade_Plc=false
End Sub
```

Delphi

```
Procedure TForm1.Button9Click(Sender: TObject);
begin
  MapPro1.Shade_Plc:=false;
end;
```

Shade_State:Boolean — Property

If set to True, the polygons defined by the State outlines are shaded. If set to false, they are not shaded, but instead assume the generic background color.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Shade_State=false
End Sub
```

Delphi

```
Procedure TForm1.Button5Click(Sender: TObject);
begin
  MapPro1.Shade_State:=false;
end;
```

ShowAllItems() — Procedure

Sets the attribute for all user-created objects to visible.

VB Example

```
Private Sub Command39_Click()
  MapPro1.ShowAllItems
End Sub
```

Delphi

```
Procedure TForm1.Button9Click(Sender: TObject);
begin
  MapPro1.ShowAllItems;
end;
```

ShowToolBar() Procedure

Displays a floating toolbar with predefined icons to automatically execute a number of the build in methods.

	Zooms the Viewport IN by a factor of 2.
	Zooms the Viewport OUT by a factor of 2.
	Reverts the Viewport to the immediately previous view.
	Zoom out so that USA extents are shown.
	Zoom out so that the world extent s are shown.
	Toggle the display of the Lon/Lat grid On/Off
	Open up the stock print dialog.
	Open up the standard routing dialog.
	Open up the Optirouter dialog (this button appears in the tool bar only if the OpriRouterBtn property is set to True).
	Open up the stock search dialog and have the ***Search for an Address*** Tab selected.
	Open up the stock search dialog and have the ***Search for a Place*** Tab selected. Note that this is also used to search four county names, either as part of a wildcard search, or by adding the suffix "County" (or "Parish" for Louisianna).
	Open up the stock search dialog and have the ***Search for an Area Code*** Tab selected .
	Open up the stock search dialog and have the ***Search for a Zip Code/Postal Code*** Tab selected.
	Open up the stock search dialog and have the ***Goto a Lon,Lat*** Tab selected.
	Invoked Windows help and opens up the file at the location specified by the developer. (Note that this is to accommodate the developer in providing help facilities for end-users of their applications, not to invoke help facilities for the OCX).
1000 mi	Sets one of the predefined Zoom scales.
-59.051136, 26.873088	This area echoes the coordinates of the current cursor position. Clicking in this area cycles the different Lon/Lat display formats available in the control.

Note: The toolbar can be cancelled by clicking on the "x" icon on the far right. When the toolbar is floating, double-clicking in the caption area anchors it (similar to setting ToolBarMode to 1). Also, left-clicking on the toolbar gives the user the option to anchor, un-anchor or close the toolbar. While the toolbar is displayed, the rightmost portion of it displays the current mouse coordinates. Clicking in the coordinate echo area cycles thought the various coordinate formats.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Parent = Form1.hdc
  MapPro1.ShowToolBar
End Sub
```

Delphi

```
Procedure TForm1.Button2Click(Sender: TObject);
begin
  MapPro1.ShowToolBar;
end;
```

StartView(s:String) — Property

A new property was added that can be set at design time. A string of the form x1,y1,x2,x2, (Upper Left, Lower Right Lat/Lon Values of Map) sets the viewport extents when the application starts up, and it takes precedence over LastView or any other viewport settings in the .CFG file.

Street:string — Property

This string holds the data returned by the Findstreet method when an ONFIND event is triggered. Data in the string consists of 8 fields delimited by ASCII character #9 and is in the order shown below. (See FindStr for more information).

```
Streetname,Cityname,Address,State,Zipcode,Lon,Lat,Distance
```

When searching for a Cross Street, the field "Streetname" contains the concatenation of both cross street names, preceded by an asterisk, and the Address fields contains "N/A".

Note: It should be emphasized that this is used only for the Street search. "Result" is used to return information from the other available search operations.

VB Example

```
Private Function MyQFactor() As Integer
MyQFactor = 1
 If MapPro1.GeoFindParse(3, MapPro1.Street) <> "" Then
   MyQFactor = 5
 Else
   If MapPro1.GeoFindParse(1, MapPro1.Street) <> "" Then
     MyQFactor = 4
   Else
     If MapPro1.GeoFindParse(2, MapPro1.Street) <> "" Then
       MyQFactor = 3
     Else
       If MapPro1.GeoFindParse(5, MapPro1.Street) <> "" Then
         MyQFactor = 2
       Else
```

```
         If MapPro1.GeoFindParse(4, MapPro1.Street) <> "" Then
MyQFactor = 1
       End If
     End If
   End If
 End If
End Function
```

Synch:Boolean — Property

Determines the rotation of street labeling when the Rotate procedure is called. If Synch is false (default), then the street label rotation is restricted in the eastern quadrants. If Synch is true, then the rotation of the street labels stays synchronized with the street segment (retains original relative orientation), around 360 degrees.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Synch=true
End Sub
```

Delphi

```
Procedure TForm1.Synch(Sender: TObject);
begin
  MapPro1.Synch:=true;
end;
```

Texture:Boolean — Property

Displays areas that do not contain map data with a gray textured surface.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Texture = True
End Sub
```

TitlePrint(s:String) — Property

Sets the title to be printed at the top of the page of the driving directions and the driving map(s) in MapPro.. Sets the footer text displayed at the bottom of maps printed with ExecPrint.

VB Example

```
Private Sub Command39_Click()
  MapPro1.TitlePrint("My Printed Directions")
End Sub
```

Delphi

```
Procedure TForm1.Button2Click(Sender: TObject);
begin
  MapPro1.TitlePrint('My Printed Directions');
end;
```

TitleUser(s:String) — Property

Sets the string to be printed as part of the footer of the driving directions pages and the driving map(s). Sets the footer text displayed at the bottom right of maps printed with ExecPrint.

VB Example

```
Private Sub Command39_Click()
  MapPro1.TitleUser("Copyright 1999 XYZ Company")
End Sub
```

Delphi

```
Procedure TForm1.Button2Click(Sender: TObject);
begin
  MapPro1.TitleUser('Copyright 1999 XYZ Company');
end;
```

ToolBarMode:Integer — Property

Sets the mode for the built-in toolbar.

It can take one of three values (also enumerated, depending on the development environment).

0 or TbHide = Hides (closes) the toolbar
1 or TbFix = Displays the toolbar fixed in the upper left corner of the form.
2 or TbFloat = Displays the toolbar floating in the general form area.

Note: Note that when the toolbar is floating, double-clicking in the caption area anchors it (similar to setting ToolBarMode to 1). Also, left-clicking on the toolbar gives the user the option to anchor, un-anchor or close the toolbar.

VB Example

```
Private Sub Command39_Click()
  MapPro1.toolbarmode=1
End Sub
```

Delphi

```
Procedure TForm1.Button2Click(Sender: TObject);
// ------------------------------------------------
// Display a fixed (anchored) toolbar
// ------------------------------------------------
begin
  MapPro1.toolbarmode:=1;
end;
```

Underlay:Boolean — Property

Controls visibility of the bitmap underlay file, if present. True state sets the underlay to visible.

Note: Underlays do NOTprint.

VB Example

```
Private Sub Command39_Click()
```

```
  MapPro1.Underlayfile="C:\pmap40\contour\pmap.pcc"
  MapPro1.underlay=true
End Sub
```

Delphi

```
Procedure TForm1.UnderlayCN1Click(Sender: TObject);
begin
  MapPro1.Underlayfile:='C:\pmap40\contour\pmap.pcc';
  MapPro1.underlay:=true;
end;
```

UnderlayFile(s:String) — Property

Specifies the name of the underlay bitmap file. You can use a wildcards "*" for the software to dynamically check all similar files and load the appropriate image if it is located in the view port. Version 7.1 of MapPro will load files of .BMP, .GIF or .JPG format. (Also see the PostUnderlay property that determines the sequence in which the underlay is drawn).

Note: A full file name specification (including path) is required, e.g., drive:\Folder\FileName.BMP. It should be emphasized that the OCX will also look in the specified directory for a file 'FileName.SAT'. This file contains the top left and bottom right Lon/Lat coordinates separated by comma. Using the OCX method ZoomUnderlay, after loading the file, will relocate the viewport to the location described by the .SAT file.

The SAT file layout is as follows: UpperLON, UpperLAT, LowerLON, LowerLAT:double

VB Example

```
Private Sub Command39_Click()
  ' Use the sample Underlay file on the Precision Mapping
  ' Streets 4.0 CD-ROM disk, assuming that the CD-ROM drive
  ' letter is D:  Note that the OCX looks for a file
  ' 'st_louis.sat' in the same directory as the
  ' 'st_louis.BMP' file.  The contents of the .SAT file, in
  ' this case, are:
  '
  '    -90.979339, 39.013707,-90.031627, 38.182347
  '
  ' which are the upper right and bottom left coordinates
  ' of the area covered by the st_louis.bmp image file}
  MapPro1.UnderlayFile="D:\pmap40\contour\st_louis.bmp"
End Sub
```

Delphi

```
Procedure TForm1.UnderlaySL1Click(Sender: TObject);
begin
  { Use the sample Underlay file on the Precision Mapping
    Streets 4.0 CD-ROM disk, assuming that the CD-ROM drive
    letter is D:  Note that the OCX looks for a file
    'st_louis.sat' in the same directory as the
    'st_louis.BMP' file.  The contents of the .SAT file, in
    this case, are:

      -90.979339, 39.013707,-90.031627, 38.182347

    which are the upper right and bottom left coordinates
    of the area covered by the st_louis.bmp image file}
```

```
  MapPro1.UnderlayFile:='D:\pmap40\contour\st_louis.bmp';
end;
```

UnderlayTransparent:Boolean — Property

Declares whether the loaded overlay will have a transparent color (see OverlayTrColor property), or not.

VB Example

```
Private Sub Command71_Click()
 'Open stock dialog to open underlay file
 CommonDialog2.ShowOpen
 s = CommonDialog2.FileName
 ' Set underlay file
 MapPro1.UnderlayFile = s
 ' Set the Underlay transparency mode
 MapPro1.UnderlayTransparent = True
 ' Set the Underlay Transparent color (white)
 MapPro1.UnderlayTrColor = RGB(255, 255, 255)
 'Redraw the Map
 MapPro1.Redraw
End Sub
```

UnderlayTrColor:Integer — Property

Defines what the transparent color in an underlay is (RGB color). This has an effect only when UnderlayTransparent =True.

VB Example

```
Private Sub Command71_Click()
 'Open stock dialog to open underlay file
 CommonDialog2.ShowOpen
 s = CommonDialog2.FileName
 ' Set underlay file
 MapPro1.UnderlayFile = s
 ' Set the Underlay transparency mode
 MapPro1.UnderlayTransparent = True
 ' Set the Underlay Transparent color (white)
 MapPro1.UnderlayTrColor = RGB(255, 255, 255)
 'Redraw the Map
 MapPro1.Redraw
End Sub
```

ViaCount:Integer — Property

Total number of via points currently defined.

ViewCmd(s:String) Property

This forces the system to open a map to a specific location upon startup. "S" is a composite string made up of that can have two different forms, depending on its first character.

If the first character is 'P', then it denotes that the viewport is to be set by a center point and a scale value, so **S** is of the form: Px,y,Scale.

If the first character is 'W', it denotes that a viewport windows will be set, so it is of the form: Wx1,y1,x2,y2. This format is also the same as the ZoomSp property.

VB Example

```
Private Sub Form_Load()
   MapPro1.ToolbarMode = TbFix
   ' Set the viewport using the Window method
   ' This was implemented to address the start-up issues
   MapPro1.ViewCmd = "w-80,45,-92,38"
End Sub
```

Visible:Boolean Property

Sets the visibility of the Map object.

VB Example

```
Private Sub Command39_Click()
  'Turn visibility off, zoom the map and turn it ON
  MapPro1.Visible=false
  MapPro1.ZoomPan(2)
  MapPro1.Zoomin
  MapPro1.Visible=true
End Sub
```

Delphi

```
Procedure TForm2.Button4Click(Sender: TObject);
{Turn visibility off, zoom the map and turn it ON}
begin
  MapPro1.Visible:=false;
  MapPro1.ZoomPan(2);
  MapPro1.Zoomin;
  OlePmMa1p.Visible:=true;
end;
```

Xcord:Double Property

Returns the Longitude of the current cursor position in decimal degrees.

VB Example

```
Private Sub Command39_Click()
  Text1.text =str(MapPro1.xcord)+ ", " + str(MapPro1.ycord)
End Sub
```

Delphi

```
Procedure TForm1.MapPro1MouseMove(Sender: TObject;
```

```
                    Shift: TShiftState; X, Y: Integer);
var xtemp,ytemp:string[12];
begin
  with MapPro1 do
  begin
    str(MapPro1.xcord:10:6,xtemp);
    str(MapPro1.ycord:10:6,ytemp);
    panel2.caption:='W:'+xtemp+'  N:'+ ytemp;
  end;
end;
```

Ycord:Double — Property

Returns the Latitude of the current cursor position in decimal degrees.

VB Example

```
Private Sub Command39_Click()
  Text1.text =str(MapPro1.xcord)+ ", " + str(MapPro1.ycord)
End Sub
```

Delphi

```
Procedure TForm1.MapPro1MouseMove(Sender: TObject;
                    Shift: TShiftState; X, Y: Integer);
var xtemp,ytemp:string[12];
begin
  with MapPro1 do
  begin
    str(MapPro1.xcord:10:6,xtemp);
    str(MapPro1.ycord:10:6,ytemp);
    panel2.caption:='W:'+xtemp+'  N:'+ ytemp;
  end;
end;
```

ZoomAll() — Procedure

Resizes map to show continental USA within the view window. The map is redrawn.

VB Example

```
Private Sub Command39_Click()
  MapPro1.ZoomAll
End Sub
```

Delphi

```
Procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
  MapPro1.ZoomAll;
end;
```

ZoomCan() — Procedure

Resizes map to show Canada within the view window. The map is redrawn.

VB Example
```
Private Sub Command39_Click()
  MapPro1.ZoomCan
  or
  MapPro1.ZoomAll
End Sub
```

Delphi
```
Procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
  MapPro1.ZoomCan;
  or
  MapPro1.ZoomAll;
end;
```

ZoomIn() Procedure

Resizes map by a factor of 2. All data required to display the map at the new size is automatically loaded (provided the path properties have been correctly defined). The map is redrawn at its new size.

VB Example
```
Private Sub Command39_Click()
  MapPro1.ZoomIn
End Sub
```

Delphi
```
Procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  MapPro1.ZoomIn;
end;
```

ZoomIWindow(x1, y1, x2, y2: Interger) Procedure

Resizes map so that rectangle specified by screen coordinates is totally visible. Largest dimension, height or width dominates.

VB Example
```
Private Sub Command39_Click()
  MapPro1.ZoomWindow(10,10,60,50)
End Sub
```

Delphi
```
Procedure TForm1.Button1Click(Sender:Object);
begin
  MapPro1.ZoomIWindow(10,10,60,50)
end;
```

ZoomLast() Procedure

Restores the previous view resulting from any view operation.

VB Example
```
Private Sub Command39_Click()
  MapPro1.ZoomLast
End Sub
```

Delphi

```
Procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
  MapPro1.ZoomLast;
end;
```

ZoomOut() — Procedure

Resizes map downwards by a factor of 1/2. All data required to display the map is automatically loaded.

VB Example

```
Private Sub Command39_Click()
  MapPro1.ZoomOut
End Sub
```

Delphi

```
Procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
  MapPro1.ZoomOut;
end;
```

ZoomOverlay() — Procedure

Calculates the extents based on the elements in the currently loaded overlay and repositions and resizes viewport so that they are ALL visible. Note that this only applies to the older overlays of the type of .CMX and .OVR, which have been loaded using the OpenOverlay method. It does *not* aply to the newer CAD overlays which have their own Zoom method through the CAD interface.

Notes There is no effect if there is no Overlay file loaded.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Openoverlay("D:\PMAP40\OVERLAYS\myfile.ovr")
  MapPro1.ZoomOverlay
End Sub
```

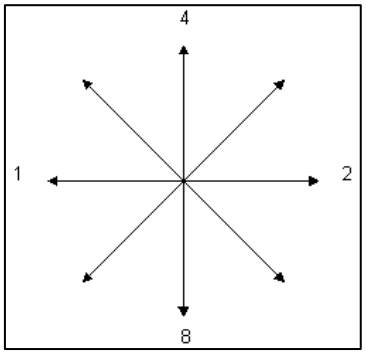
Delphi

```
Procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
  MapPro1.Openoverlay('D:\PMAP40\OVERLAYS\myfile.ovr');
  MapPro1.ZoomOverlay;
end;
```

ZoomPan(i:Integer) — Procedure

Specified number causes viewport to move by a fixed amount in each of 8 principal directions. The directions are defined as shown below:

```
Direction    S  N   E   W
Bit          8  4   2   1
```



To move north, specify a ZoomPan integer factor of "4". To move west, specify 1. To move NorthEast use 6 (2+4), to move SouthWest use 9, with SouthEast use 10, etc.

VB Example

```
Private Sub Command39_Click()
  'Pan the viewport NorthWest
  MapPro1.ZoomPan(5)
End Sub
```

Delphi

```
Procedure TForm1.Button2Click(Sender: TObject);
begin
  {Pan the viewport NorthWest}
  MapPro1.ZoomPan(5);
end;
```

ZoomSP(s:String) **Procedure**

This Zooms the system to open a map to a specific location. "S" is a composite string made up of that can have two different forms, depending on its first character.

If the first character is 'P', then it denotes that the viewport is to be set by a center point and a scale value, so **S** is of the form: Px,y,Scale.

If the first character is 'W', it denotes that a viewport windows will be set, so it is of the form: Wx1,y1,x2,y2. This format is also the same as the ViewCmd property.

VB Example

```
Private Sub Form_Load()
   MapPro1.ToolbarMode = TbFix
   ' Set the viewport using the Window method
   ' This was implemented to address the start-up issues
   MapPro1.ZoomSp("w-80,45,-92,38")
End Sub

Private Sub Command1_Click()
   ' Set the viewport using the Point method
   MapPro1.ZoomSP "P-82.1234,41.4321,15"
End Sub
```

ZoomUnderlay() Procedure

Repositions and resizes viewport so that the area defined by the 'top left' and 'bottom right' coordinated in the '.SAT' file associated with the currently loaded Underlay file, is visible.

VB Example

```
Private Sub Command39_Click()
  MapPro1.UnderlayFile="D:\pmap40\contour\st_louis.bmp"
  MapPro1.ZoomUnderlay
End Sub
```

Delphi

```
Procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
  MapPro1.UnderlayFile:='D:\pmap40\contour\st_louis.bmp';
  MapPro1.ZoomUnderlay;
end;
```

ZoomWindow(x1, y1, x2, y2: OleVariant) Procedure

Resizes map so that rectangle specified by LON/LAT coordinates (decimal degrees) is totally visible. Largest dimension, height or width dominates. Resizes map so that rectangle specified (in decimal degrees) is visible. All data required to display map is automatically loaded. The order of the points (top-left, bottom-right, etc. is not important as they are ordered by the OCX.

Note: It should be noted that the actual extents of the viewport will not necessarily be those specified by the user, as such extents depend on the current viewport size. The largest dimension (height/width) dominates. However, the centroid of the requested windows will be located in the center of the view port.

VB Example

```
Private Sub Command39_Click()
  MapPro1.Zoomwindow(-120,30,-100,40)
End Sub
```

Delphi

```
Procedure TForm1.Button11Click(Sender: TObject);
begin
  MapPro1.Zoomwindow(-120,30,-100,40);
end;
```

.CAD Interface

An interface that enables the user to annotate a map surface by drawing desired objects on a "user" layer, superimposed on the map. These objects and their attributes can be set and edited either programmatically, or through the built-in toolbars, off the .CAD interface, as described later in this section. Objects are stored in an indexed array and can be randomly accessed and modified through that array. The map control needs to be refreshed, following the creation of these objects, for them to be rendered on the map surface.

User drawn objects can be saved to an external binary file (with the default extension *.CAD*), and can then be loaded from such a file either replacing any currently defined objects or being appended to them.

All objects defined in the CAD interface, have the following properties/methods.

.*CAD.Object*.**Brush**
.*CAD.Object*.**Caption**
.*CAD.Object*.**Font**
.*CAD.Object*.**GreatCircle**
.*CAD.Object*.**MoveAbs**
.*CAD.Object*.**MoveRel**
.*CAD.Object*.**Objecttype**
.*CAD.Object*.**Pen**
.*CAD.Object*.**Selected**
.*CAD.Object*.**Tag**
.*CAD.Object*.**Visible**

Details of these properties methods are included at the end of the CAD interface definition. New mdCAD mode.

CAD objects can be created in a number of ways.

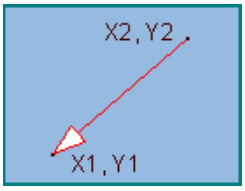
(a) Programatically, by directly calling a CAD.*Object* method, and specifying all the appropriate parameters.

(b) By setting the appropriate mapMode value (mapMode=mdCAD), and the desired CAD.ObjectType, and then interactively drawing the object specified by ObjectType, or

(c) By opening the CAD toolbar and clicking the appropriate button to draw the desired object.

.CAD.Arrow(x1,y1,x2,y2:Double):CadObj Method

Creates an Arrow object. X1,X2 are the Lon/Lat coordinates of the arrowhead, and X2,Y2 the coordinates of the arrow tail endpoint.



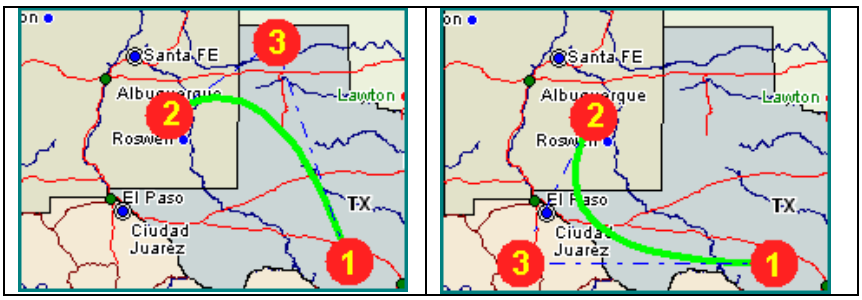
The arrow object may be re-sized, moved, etc, once it has been created, by invoking the CAD Edit mode from the CAD toolbar, os setting ObjectType=0 and then setting mapMode to mdCAD.

VB Example

```
Private Sub Command51_Click()
Dim Arr As CadObj
  ' Draw the arrow - second point is tip location
  Set Arr = MapPro1.Cad.Arrow(-100, 30, -110, 40)
  ' Set some brush attributes
  Arr.Brush.Color = vbBlue
  Arr.Brush.BackColor = vbGreen
  Arr.Brush.Style = 4
  Arr.Brush.Mode = 2 ' 1 would be transparent
  ' Set some pen attributes
  Arr.Pen.Width = 4
  Arr.Pen.Color = vbRed
  ' Set some Caption Attributes
  Arr.Caption = "Sample Arrow Object"
  Arr.Font.Height = 14
  Arr.Font.Style = 1 'bold
  MapPro1.Refresh
End Sub
```

.CAD.Bezier(X1,Y1,X2,Y2,X3,Y3:Double):CadObj Method

Creates a bezier line object, using the coordinates of the three specified points, as shown in the graphics below.



VB Example

```
Private Sub Command52_Click()
Dim Bz As CadObj, Pt1 As CadObj, Pt2 As CadObj, Pt3 As CadObj
```

```
Dim Ln1 As CadObj, Ln2 As CadObj
  MapPro1.Cad.Clear
  ' Set the coordinates for the three points
  Pt1X = -100
  Pt1Y = 30
  Pt2X = -105
  Pt2Y = 34
  Pt3X = -107
  Pt3Y = 30
  Set Bz = MapPro1.Cad.Bezier(Pt1X, Pt1Y, Pt2X, Pt2Y, Pt3X, Pt3Y)
  ' Show the Bezier control lines
  Set Ln1 = MapPro1.Cad.mLine(Pt1X, Pt1Y, Pt3X, Pt3Y)
  Set Ln2 = MapPro1.Cad.mLine(Pt2X, Pt2Y, Pt3X, Pt3Y)
  Ln1.Pen.Style = 3
  Ln1.Pen.Color = vbBlue
  Ln2.Pen.Style = 3
  Ln2.Pen.Color = vbBlue
  ' The commented alternative would mark the 3 points
  ' by getting and then using the handles to the built-in markers.
  '   s1 = MapPro1.Cad.GetMarker(2)
  '   s2 = MapPro1.Cad.GetMarker(2)
  '   s2 = MapPro1.Cad.GetMarker(7)
  ' We'll use external bitmaps, instead
  Set Image1.Picture = LoadPicture("d:\test\mappro71\One.gif")
  Set Image2.Picture = LoadPicture("d:\test\mappro71\Two.gif")
  Set Image3.Picture = LoadPicture("d:\test\mappro71\Three.gif")
  Set Pt1 = MapPro1.Cad.Marker(Pt1X, Pt1Y, Image1.Picture.Handle)
  Set Pt2 = MapPro1.Cad.Marker(Pt2X, Pt2Y, Image2.Picture.Handle)
  Set Pt3 = MapPro1.Cad.Marker(Pt3X, Pt3Y, Image3.Picture.Handle)
  ' Set some attributes
  Bz.Pen.Width = 4
  Bz.Pen.Color = vbGreen
  MapPro1.Refresh
End Sub
```

.CAD.BringToFront — Method

Brings the currently selected CAD objects to the front, i.e., draws them after (on top of) all other CAD Objects. See the SelectRange and SelectRect methods for selecting CAD objects to operate on.

VB Example

```
Private Sub Command58_Click()
Dim Bz As CadObj, Pt1 As CadObj, Pt2 As CadObj, Pt3 As CadObj
Dim Ln1 As CadObj, Ln2 As CadObj, rc1 As CadObj, rc2 As CadObj, rc3
As CadObj
  MapPro1.Cad.Clear
  ' Set the coordinates for the three points
  Pt1X = -100
  Pt1Y = 30
  Pt2X = -105
  Pt2Y = 34
  Pt3X = -102
  Pt3Y = 36
  s1 = MapPro1.Cad.GetMarker(2)
  s2 = MapPro1.Cad.GetMarker(4)
  s3 = MapPro1.Cad.GetMarker(7)
```

```
  Set Pt1 = MapPro1.Cad.Marker(Pt1X, Pt1Y, s1)
  Set Pt2 = MapPro1.Cad.Marker(Pt2X, Pt2Y, s2)
  Set Pt3 = MapPro1.Cad.Marker(Pt3X, Pt3Y, s3)
  Pt1.Caption = "point[1]"
  Pt1.Font.Align = 1
  Pt2.Caption = "point[2]"
  Pt2.Font.Align = 2
  Pt3.Caption = "point[3]"
  ' create a circle and an ellipse
  Set rc1 = MapPro1.Cad.mCircle(-100, 30, -80, 38, 1)
  rc1.Caption = "Circle[4]"
  rc1.Brush.Style = 4
  rc1.Brush.Mode = 2
  Set rc2 = MapPro1.Cad.Ellipse(-99, 28, -92, 40)
  rc2.Caption = "Ellipse[5]"
  rc2.Brush.Color = vbBlue
  'Also draw enclosing polygon
  Set rc3 = MapPro1.Cad.Rectangle(-99, 28, -92, 40)
  rc3.Brush.Style = 1
  ' Bring the first object Tofront, i.e. above the other objects
  With MapPro1.Cad
    .Objects(0).Selected = True
    .BringToFront
  End With
  MapPro1.Refresh
End Sub
```

.CAD.Brush — Property

Sets the default brush properties for all CAD objects. When an object is created it inherits these brush properties, unless specific brush properties are set for the object, either prior to, or after its creation. (See *object.brush*, later on, for a detailed description of the brush properties)

VB Example

```
Private Sub Command41_Click()
Dim RegPol As CadObj
' Set default Brush Properties for all CAD objects
  MapPro1.Cad.Brush.Color = vbRed
  MapPro1.Cad.Brush.BackColor = vbBlue
  MapPro1.Cad.Brush.Mode = vbTransparent
  MapPro1.Cad.Brush.Style = vbCross
  ' Draw a polygon to test settings
  Set RegPol = MapPro1.Cad.RegularPolygon(-100, 40, -80, 40, 4)
  ' refresh the map to see object
  MapPro1.Refresh
End Sub
```

.CAD.Clear() — Method

Clears all CAD objects currently defined in memory. Note that unless you have saved your CAD objects to an external file, you will not be able to recover them following the issuance of .Clear command. Also, remember to refresh the map object following the .Clear call.

VB Example

```
Private Sub Command38_Click()
' Clear all CAD objects, but first save them to an external file
  MapPro1.Cad.SaveToFile ("MyTestCadFile")
  MapPro1.Cad.Clear
  MapPro1.Refresh
End Sub
```

.CAD.Count:Integer — Property

Holds the count of the total CAD objects currently defined in memory. Note that the Objects array is zero-based, so if *CAD.Count = 5*, for example, the Objects() array is represented by Object(0)..Object(4).

VB Example

```
Private Sub Command42_Click()
' List the types of all current objects  (note array is zero based)
  List1.Clear
  For i = 1 To MapPro1.Cad.Count
    List1.AddItem "Item # " & i & ", Type: " & MapPro1.Cad.Objects(i
- 1).ObjectType
  Next i
End Sub
```

.CAD.Delete():Integer — Method

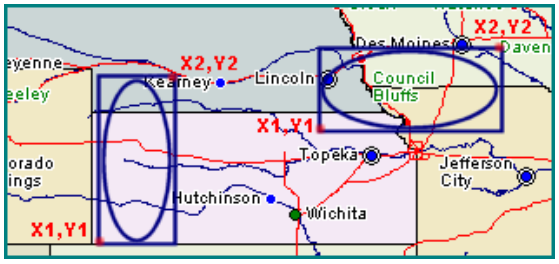
Deletes the currently selected CAD objects. See the *SelectRange* and *SelectRect* methods for selecting CAD objects to operate on. It returns the number of CAD objects that were deleted.

VB Example

```
Private Sub Command55_Click()
  ' Delete all selected objects and display the # of
  ' objects deleted (for confirmation purposes)
  n = MapPro1.Cad.Delete
  Command55.Caption = "Last Delete=" & n
End Sub
```

.CAD.Ellipse(X1, Y1, X2, Y2:Double):CadObj — Method

Creates an ellipse within the bounding rectangle defined by the two points **X1,Y1** and **X2,Y2.**



VB Example

```
Private Sub Command25_Click()
Dim TEL As CadObj, Trec As CadObj
   ' Draw the bounding rectangle
   Set Trec = MapPro1.Cad.Rectangle(-120, 40, -82, 30)
   Trec.Pen.Style = 3
   Set TEL = MapPro1.Cad.Ellipse(-120, 40, -82, 30)
   ' Set some of the line attributes
   TEL.Pen.Width = 1
   TEL.Pen.Color = vbBlue
   TEL.Pen.BackColor = vbRed
   TEL.Pen.Style = 1
   'TEL.Pen.Mode = 0
   TEL.Brush.Color = vbGreen
   TEL.Brush.BackColor = vbYellow
   TEL.Brush.Style = Val(Text18.Text) ' I varried these 0-5
   TEL.Brush.Mode = Val(Text19.Text) ' Varried these 0-5
   MapPro1.Refresh
End Sub
```

.CAD.Extents :TExtentRec — Method

A record containing the extents of all current CAD objects. This can be used with ZoomWindow to zoom to zoom in or out to the extents of existing CAD objects. The record contains 4 doubles, as shown below:

```
TExtentRec = record
              Xmin:double;
              Ymin:double;
              Xmax:double;
              Ymax:double;
             end;
```

VB Example

```
Private Sub Command54_Click()
' Use the CAD. Extents to zoom to all the CAD objects
With MapPro1.Cad.Extents
  MapPro1.ZoomWindow .Xmin, .Ymin, .Xmax, .Ymax
End With
End Sub
```

.CAD.Font() — Method

Sets the default font properties for all CAD objects (the caption of the object). When an object is created it inherits these font properties, unless specific font properties are set for the object, either prior to, or after its creation. (See *object.font* later on for a detailed description of the brush properties)

VB Example

```
Private Sub Command31_Click()
```

```
  ' Set Text Default Attributes
    Mappro1.CAD.Font.Height = 42
    Mappro1.CAD.Font.Mode = -1
    Mappro1.CAD.Font.Angle = 30
  End Sub
```

.CAD.GetMarker(n:Integer):LongInteger Method

Returns a handle to the n-th marker in the built-in markers array, which can then be used to place a marker on the map using the CAD.Marker method.

VB Example

```
Private Sub Command28_Click()
Dim Tmk1 As CadObj, Tmk2 As CadObj
  ' Get the handle of one of the built-in markers
  hmark = MapPro1.Cad.GetMarker(2)
  ' Now use that handle to paint the marker on the map
  Set Tmk1 = MapPro1.Cad.Marker(-92, 27, hmark)
  ' Select a user-defined bitmap to use as a marker
  Set Image1.Picture = LoadPicture("d:\test\mappro71\test.bmp")
  Set Tmk2 = MapPro1.Cad.Marker(-72, 42, Image1.Picture.Handle)
  'Refresh the map
  MapPro1.Refresh
End Sub
```

.CAD.GetMetaObj(s:String):LongInteger Method

Returns a handle to the windows metafile specified by the user, primarily to be used with the MetaObj object. This method works both with metafiles and enhanced metafiles.

VB Example

```
Private Sub Command60_Click()
Dim Wm1 As CadObj, Mhnd As Long
  ' Get handle to user's metafile
  Mhnd = MapPro1.Cad.GetMetaObj("d:\test\mappro71\One.wmf")
  ' Use handle to set MetaObj
  Set Wm1 = MapPro1.Cad.MetaObj(-100, 30, -80, 36, Mhnd)
  MapPro1.Refresh
End Sub
```

.CAD.GetSymbol(index:Integer):LongInteger Method

Returns a handle to the built-in symbols windows metafile specified by the user, primarily to be used with the Symbol object. The vailable range of values for index are 1..95.

VB Example

```
Private Sub Command60_Click()
Dim Wm1 As CadObj, Mhnd As Long
  ' Get handle to built-in symbol
  Mhnd = MapPro1.Cad.GetSymbol(5)
  ' Use handle to set Symbol
```

```
  Set Wm1 = MapPro1.Cad.Symbol(-100, 30, -80, 36, Mhnd)
  MapPro1.Refresh
End Sub
```

.CAD.Group() Method

Groups the currently selected CAD objects into a composite object that can then be modified as a single entity. Grouped CAD objects can be broken down to the components using the ***.CAD.Ungroup*** method.

VB Example

```
Private Sub Command66_Click()
  ' Ungroup selected COmplex Object
  MapPro1.Cad.UnGroup
  ' Update to show new objecs.  Mode to show handles.
  MapPro1.MapMode = MdCad
  MapPro1.Cad.ObjectType = 0
  MapPro1.Refresh
End Sub
```

.CAD.ImportFile() Method

Imports a CMX or OVR file (older overlay file format), and appends the objects to any objects currently defined. Note that at the end of the Import process, the elements in memory have been converted to the native CAD format and can be modified using the CAD interface. This is different than the OpenOverlay method which simply opens the older overlay files without altering their format.

VB Example

```
Private Sub Command108_Click()
  ' Import an older-type OVR overlay file
  ' Objects in it get converted to CAD objects
  MapPro1.Cad.ImportFile "d:\test\mappro71\desmgrms.ovr"
  ' MapPro1.OpenOverlay "d:\test\mappro71\desmgrms.ovr"
  ' Zoom to the extents of the Overlay
  MapPro1.ZoomOverlay
End Sub
```

.CAD.LoadFromFile(s:String; Option:Integer) Method

Loads the CAD object in the specified external file. Note: the map control needs to be refreshed, following the file loading, for the CAD objects to be painted on the map surface. If **Option=0**, then the existing CAD layer is cleared before the file is loaded. If **Option=1**, then the CAD objects in the file are appended to any CAD objects currently defined.

VB Example

```
Private Sub Command33_Click()
  ' Load CAD objects from file (Clear objects first)
  MapPro1.Cad.LoadFromFile "myCadTest01.cad",0
  ' refresh the map to see the objects
  MapPro1.Refresh
End Sub
```

.CAD.Marker(X, Y:Double; hnd:Integer) Method

Creates an instance of the marker (bitmap) defined by the handle (hnd) at the specified Lon, Lat coordinates. The handle can be that of one of the built-in bitmap markers (see *CAD.GetMarker*), or one externally defined by the user.

VB Example

```
Private Sub Command28_Click()
Dim Tmk1 As CadObj, Tmk2 As CadObj
  ' Get the handle of one of the built-in markers
  hmark = MapPro1.Cad.GetMarker(2)
  ' Now use that handle to paint the marker on the map
  Set Tmk1 = MapPro1.Cad.Marker(-92, 27, hmark)
  ' Select a user-defined bitmap to use as a marker
  Set Image1.Picture = LoadPicture("d:\test\mappro71\test.bmp")
  Set Tmk2 = MapPro1.Cad.Marker(-72, 42, Image1.Picture.Handle)
  'Refresh the map
  MapPro1.Refresh
End Sub
```

.CAD.mCircle(Xc, Yc, Xp, Yp, Aspect:Double):CadObj Method

Returns a CadObj structure, and draws a circle given the following information:

Xc,Yc Lon, Lat coordinates (in decimal degrees) of the center of the circle.

Xp,Yp Lon, Lat coordinates of the first point on the circle circumference.
Note that if Xp=Xc then Yp-Yc is the radius of the circle in degrees, and if Yp=Yc, then Xp-Xc is the radius of the circle in degrees,

Aspect The circle aspect ratio (horizontal to vertical diameter ratio)

VB Example

```
Private Sub Command35_Click()
Dim Circ As CadObj
  Set Circ = MapPro1.Cad.mCircle(-80, 40, -60, 40, 1.5)
  'Set some properties
  Circ.Brush.Color = vbRed
  Circ.Brush.BackColor = vbYellow
  Circ.Brush.Style = vbDash
  Circ.Brush.Mode = 0
  Circ.Pen.Color = vbGreen
  Circ.Pen.Style = vbDash
  'Refresh the map
  MapPro1.Refresh
End Sub
```

.CAD.MetaObj(X1,Y1,X2,Y2:Double; Hnd:LongInteger):CadObj Method

Draws a symbol within the specified bounding rectangle, using the handle to an external metafile specified by the user. The metafile is automatically scaled to fit within the bounding rectangle, and is then played on the screen.

VB Example

```
Private Sub Command20_Click()
Dim hndl As Long, MObj As CadObj
  ' Get handle to external meta file
  MapPro1.Cad.GetMetaObj("d:\test\mappro71\One.wmf")
  ' USe handle to create object
  Set MObj = MapPro1.Cad.MetaObj(-100, 30, -120, 41, hndl)
  ' No attributes can be changed for these objects
  MapPro1.Refresh
End Sub
```

.CAD.mLine(X1, Y1, X2, Y2:Double):CadObj — Method

Draws a line between the two specified points. The line is either a straight line or a Great Circle line depending on the GreatCircle flag of the object.

X1,Y1 Lon, Lat coordinates of the starting point

X2,Y2 Lon, Lat coordinates of the end point

VB Example

```
Private Sub Command24_Click()
Dim TL As CadObj
   ' Draw a Straight Line
   Set TL = MapPro1.Cad.mLine(-89, 32, -112, 42)
   ' Set some of the line attributes
   TL.Pen.Width = Val(Text17.Text)
   TL.Pen.Color = Val(Text7.Text)
   TL.Pen.BackColor = Val(Text8.Text)
   TL.Pen.Style = Val(Text11.Text) 'or vbDash
   Label6.Caption = MapPro1.Cad.Count
   MapPro1.Refresh
End Sub
```

.CAD.Objects(n:integer) array of CadObj — Method

An indexed array holding all the currently defined CAD objects. Each object can be directly accessed, adited, etc. by its index number. This is a zero-based array. The maximum number of objects that can be accommodated is only limited by system resources.

VB Example

```
Private Sub Command42_Click()
' List the types and pen colors of all current objects  (note array
is zero based)
  List1.Clear
  With MapPro1.Cad
    For i = 1 To .Count
      List1.AddItem "Item # " & i & ", Type: " & .Objects(i -
1).ObjectType & ", Color: " & .Objects(i - 1).Pen.Color
    Next i
```

```
    ' Now let's change the pen color of the 3rd item and list them
again
    ' this is a zero-based array, so the third item is index #2
    .Objects(2).Pen.Color = 11
    For i = 1 To .Count
      List1.AddItem "Item # " & i & ", Type: " & .Objects(i -
1).ObjectType & ", Color: " & .Objects(i - 1).Pen.Color
    Next i
  End With
End Sub
```

.CAD.ObjectType:Integer — Property

Sets the default object type, for interactive object creation. This is one of the ways the user may implement interactive object creation (the other is by invoking the *CAD.Toolbar*).

The user needs to set whatever default CAD Object attributes they want, set the desired CAD.ObjectType and then set the mapMode to mdCAD. The control goes into the interactive drawing mode until the completion of the object, and then refreshes the screen. (Note if ObjectType=0, then the control is in CAD selection mode, and if ObjectType=-1, then the control has finished its previous operation and is in NUL mode, i.e., mapMode can be safely set to mdZoom.

The available ObjectType values are:

#	**Object Type**
1	Line
2	Rectangle
3	Ellipse
4	Polyline
5	Polygon
6	Marker
7	Text
8	Circle
9	Regular Polygon
10	Free Hand (*)
11	Arrow
12	Bezier
13	Symbol
14	Text Bubble
15	Grouped Object
16	MetaObj

VB Example

```
Private Sub Command57_Click()
Dim Otp As Integer
  ' Get the Object type specified by the user
  Otp = Val(Text20.Text)
  ' Make sure it is a valid type, otherwise pop up message
  ' Remember ObjectType=-1 is NUL mode and
```

```
        ' ObjectType=0 is Select mode
        If (Otp > -2) And (Otp < 14) Then
          ' Set a global Variable flag to indicate that the control is
          ' in Programmatic Drawing Mode
          IamDrawing = True
          MapPro1.Cad.ObjectType = Otp
          MapPro1.Cad.ObjectType = Val(Text20.Text)
          MapPro1.MapMode = MdCad
          ' Note: The CADChange event needs to be checked to make sure
it's safe to
          ' set MapMode back to mdZoom
        Else
          MsgBox "Sorry, valid Object Types are 1 - 13.", vbOKOnly, "An
Error was encountered!"
          ' Reset the mode back to mdZoom
          MapPro1.MapMode = MdZoom
        End If
      End Sub


      Private Sub MapPro1_CadChange(ByVal Current As Long)
      ' Check to see if Current (ObjectType) is -1 and change MapMode
      ' Also make sure the control was in Programmatic DRAW mode
        If ((Current = -1) And (IamDrawing = True)) Then
          MapPro1.MapMode = MdZoom
          IamDrawing = False
        Else
        End If
        Beep
      End Sub
```

.CAD.Pen() Method

Sets the default pen properties for all CAD objects. When an object is created it inherits these pen properties, unless specific pen properties are set for the object, either prior to, or after its creation. (See object.pen later on for a detailed description of the brush properties)

VB Example

```
Private Sub Command44_Click()
Dim Robj As CadObj
' Set some default pen properties for the CAD Interface
  MapPro1.Cad.Pen.Color = vbRed
  MapPro1.Cad.Pen.Width = 4
  ' Create a regular polygon
  Set Robj = MapPro1.Cad.RegularPolygon(-100, 40, -100, 42, 7)
  'change pen color
  Robj.Pen.Color = vbBlue
  'Refresh the map
  MapPro1.Refresh
End Sub
```

.CAD.Polygon(Points:TRPoint; N:Long):CadObj Method

Draws a polygon object of **N** vertices. **Points** is the staring element in a TRPoints array. This allows the user to use the same array of points to define a mumber of different polygons. These are all closed polygons.

VB Example

```
Private Sub Command27_Click()
Dim Tpt(20) As TrPoint
Dim Tpg1 As CadObj, Tpg2 As CadObj
   ' Define some points to be used for polygons
   Tpt(1).x = -100
   Tpt(1).y = 40
   Tpt(2).x = -92
   Tpt(2).y = 41
   Tpt(3).x = -88
   Tpt(3).y = 32
   Tpt(4).x = -96
   Tpt(4).y = 38
   Tpt(5).x = -84
   Tpt(5).y = 39
   Tpt(6).x = -80
   Tpt(6).y = 38.5
   Tpt(7).x = -85
   Tpt(7).y = 34
   ' Draw Polygon with first 4 Points
   Set Tpg1 = MapPro1.Cad.Polygon(Tpt(1), 4)
   Tpg1.Pen.Width = 2
   Tpg1.Pen.Color = vbRed
   Tpg1.Brush.Color = vbBlue
   ' Draw a second polygon with Points 5, 6, and 7
   Set Tpg2 = MapPro1.Cad.Polygon(Tpt(5), 3)
   Tpg2.Pen.Width = 1
   Tpg2.Pen.Style = vbDash
   Tpg2.Pen.Color = vbGreen
   Tpg2.Brush.Color = vbRed
   MapPro1.Refresh
End Sub
```

.CAD.Polyline(Points:TRPoint; N:Long):CadObj Method

Draws a polyline object of **N** vertices. **Points** is the staring element in a TRPoints array. This allows the user to use the same array of points to define a number of different polylines.

VB Example

```
Private Sub Command26_Click()
Dim Tpt(10) As TrPoint
Dim Tpg As CadObj, Tpg2 As CadObj
   ' Define some points
   Tpt(1).x = -101
   Tpt(1).y = 32
   Tpt(2).x = -102
   Tpt(2).y = 34
   Tpt(3).x = -103
   Tpt(3).y = 31
   Tpt(4).x = -116
   Tpt(4).y = 40
   Tpt(5).x = -113
```

```
    Tpt(5).y = 38
    Tpt(6).x = -86
    Tpt(6).y = 34
    ' Draw Polyline with first 4 Points
    Set Tpg = MapPro1.Cad.Polyline(Tpt(1), 4)
    ' Set some of the line attributes
    Tpg.Pen.Width = 6
    Tpg.Pen.Color = vbRed
    Tpg.Caption = "This is the first Polyline"
    'Draw another line using points 2 through 6
    Set Tpg2 = MapPro1.Cad.Polyline(Tpt(2), 5)
    Tpg2.Pen.Width = 2
    Tpg2.Pen.Color = vbYellow
    MapPro1.Refresh
End Sub
```

.CAD.Rectangle(X1, Y1, X2, Y2:Double):CadObj — Method

Creates a rectangle object defined by the Lon/Lat coordinates of two opposite corners.

VB Example

```
Private Sub Command46_Click()
Dim RecO1 As CadObj, RecO2 As CadObj
' Create a rectangle Object
  Set RecO1 = MapPro1.Cad.Rectangle(-100, 30, -80, 40)
  RecO1.Pen.Color = vbRed
  'Refresh the map surface
  MapPro1.Refresh
End Sub
```

.CAD.RegularPolygon(Xc, Yc, Xp, Yp:Double; N:Integer):CadObj — Method

Creates a regular polygon Object, where…

Xc,Yc Lon, Lat coordinates (in decimal degrees) of the center of the regular polygon

Xp,Yp Lon, Lat coordinates of the first point on the circumference of the prescribed circle i.e., the virst vertex.

N The number of sides (and vertices)

VB Example

```
Private Sub Command34_Click()
Dim RegPol As CadObj
  Set RegPol = MapPro1.Cad.RegularPolygon(-100, 40, -80, 40, 9)
  RegPol.Caption = "Sample Regular Polygon"
  RegPol.GreatCircle = False
  RegPol.Pen.Color = vbBlue
  RegPol.Pen.BackColor = vbRed
  RegPol.Pen.Style = vbDash
  RegPol.Brush.Color = vbGreen
  RegPol.Brush.BackColor = vbWhite
```

```
  ' RegPol.Brush.Style = Val(Text18.Text)
  ' RegPol.Brush.Mode = Val(Text19.Text)
  ' RegPol.Pen.Color = Val(Text7.Text)
  ' RegPol.Brush.Color = Val(Text9.Text)
  ' RegPol.Brush.Mode = mergepaint
  ' RegPol.Brush.Style = Val(Text12.Text)
  MapPro1.Refresh
End Sub
```

.CAD.Rotate(Xp, Yp, Angle:Double) — Method

Rotates any selected objects through an angle specified by ***Angle***, about a point specified by ***Xp, Yp***.

VB Example

```
Private Sub Command63_Click()
Dim mk As CadObj, nhd As Integer, xt1, yt1, ang As Double
  ' Set the rotation point and Angle
  xt1 = -100
  yt1 = 20
  ang = 15
  ' Get marker and mark center of rotation point
  nhd = MapPro1.Cad.GetMarker(6)
  Set mk = MapPro1.Cad.Marker(xt1, yt1, nhd)
  ' Select the second CAD object
  MapPro1.Cad.Objects(1).Selected = True
  ' Rotate the object
  MapPro1.Cad.Rotate xt1, yt1, ang
  MapPro1.Refresh
End Sub
```

.CAD.SaveToFile(S:String) — Method

Saves all currently defined CAD objects to a binary CAD file specified by **S**. It shuld be pointed out that when an application using the OCX is started, if a file named "AUTOLOAD.CAD" is found in the applications directory, it will be automatically loaded.

VB Example

```
Private Sub Command32_Click()
  ' Save CAD objects to file
  MapPro1.Cad.SaveToFile "myCadTest02"
  ' Clear all Objects from Map
  MapPro1.Cad.Clear
  'Refresh the map
  MapPro1.Refresh
End Sub
```

.CAD.ScaleToSize(Value,Height:Double):Double Function

This CAD helper function takes **Value**, which is the map scale in miles/km that you are interested it, and Height, which is the font heigh (in pixels) that you would like at **Value** scale, and returns the Size that you should be passing to TextFloat CAD method.

Delphi Example

```
procedure TForm1.Button17Click(Sender: TObject);
var tmk1, tmk2:array [1..10] of CadObj;
   i:integer;
   x:double;
begin
  // Set 2 TextFloat and 2 Text objects
  For i := 1 To 2 do
  begin
   // Get the value to use for 12 pixel height at 5 mile scale
   x:=mappro1.cad.scaletosize(5,12);
   tmk1[i] := MapPro1.Cad.Textfloat(-110 + (i * 0.02),
          30 + (i * 0.02), x, 'Item: ' + inttostr(i));
   tmk1[i].Font.color := clblue;
   tmk2[i] := MapPro1.Cad.Text(-110 - (i * 0.04),
          30 + (i * 0.04), 'Item: ' + inttostr(i));
  end;
  mappro1.GotoPoint(-110,30);
  mappro1.miles:5;
end;
```

.CAD.SelectRange(Ns,Nn,Option:Integer) Method

Sets the ***.Selected*** property of a user specified number of CAD objects.

Ns – The starting index # of the Objects to be acted on

Nn – The Last index # of the object to be acted on

Option – It can have the following values:

0 – De-Select all objects in Selection rectangle, i.e. set their .Selected property to False.
1 – Select all objects in Selection rectangle, i.e. set their .Selected property to True
2 – Toggle the .Selected property of all objects in the selection rectangle.

VB Example

```
Private Sub Command52_Click()
Dim Bz As CadObj, Pt1 As CadObj, Pt2 As CadObj, Pt3 As CadObj
```

```
Dim Ln1 As CadObj, Ln2 As CadObj, Sb1 As CadObj
  MapPro1.Cad.Clear
  ' Set the coordinates for the three points
  Pt1X = -100
  Pt1Y = 30
  Pt2X = -105
  Pt2Y = 34
  Pt3X = -107
  Pt3Y = 30
  Set Bz = MapPro1.Cad.Bezier(Pt1X, Pt1Y, Pt2X, Pt2Y, Pt3X, Pt3Y)
  ' Show the Bezier control lines
  Set Ln1 = MapPro1.Cad.mLine(Pt1X, Pt1Y, Pt3X, Pt3Y)
  Set Ln2 = MapPro1.Cad.mLine(Pt2X, Pt2Y, Pt3X, Pt3Y)
  Ln1.Pen.Style = 3
  Ln1.Pen.Color = vbBlue
  Ln2.Pen.Style = 3
  Ln2.Pen.Color = vbBlue
  ' We'll use external bitmaps
Set Image1.Picture = LoadPicture("d:\test\mappro71\One.gif")
  Set Image2.Picture = LoadPicture("d:\test\mappro71\Two.gif")
  Set Image3.Picture = LoadPicture("d:\test\mappro71\Three.gif")
  Set Pt1 = MapPro1.Cad.Marker(Pt1X, Pt1Y, Image1.Picture.Handle)
  Set Pt2 = MapPro1.Cad.Marker(Pt2X, Pt2Y, Image2.Picture.Handle)
  Set Pt3 = MapPro1.Cad.Marker(Pt3X, Pt3Y, Image3.Picture.Handle)
  ' Set some attributes
  Bz.Pen.Width = 4
  Bz.Pen.Color = vbGreen
  Set Sb1 = MapPro1.Cad.mCircle(-104, 35, -120, 38, 1)
  ' Select the seventh and send place it underneath the other
objects
  n = MapPro1.Cad.SelectRange(6, 7, 1)
  MapPro1.Cad.SendToBack
  MapPro1.Refresh
End Sub
```

.CAD.SelectRect(x1,y1,x2,y2:Double, Option:Integer) Method

Sets the selected flag of any object within the specified rectangle.

X1,Y1 – Lon/Lat coordinates of one corner of Selection rectangle

X2,Y2 – Lon/Lat coordinates of opposite corner of Selection rectangle

Option – It can have the following values:

0 – De-Select all objects in Selection rectangle, i.e. set their .Selected property to False.
1 – Select all objects in Selection rectangle, i.e. set their .Selected property to True
2 – Toggle the .Selected property of all objects in the selection rectangle.

VB Example `Private Sub Command56_Click()`

```
Dim r1 As CadObj
  Set r1 = MapPro1.Cad.Rectangle(-70, 32, -120, 46)
  r1.Brush.Style = 1
  n = MapPro1.Cad.SelectRect(-70, 32, -120, 46, 1)
  Command56.Caption = "Selected =" & n
  MapPro1.Refresh
End Sub
```

.CAD.SendToBack Property

Sends any currently selected objects to the back, i.e., they are drawn before (underneath) any other CAD objects. See the SelectRange and SelectRect methods for selecting CAD objects to operate on.

VB Example

```
Private Sub Command53_Click()
Dim Bz As CadObj, Pt1 As CadObj, Pt2 As CadObj, Pt3 As CadObj
Dim Ln1 As CadObj, Ln2 As CadObj, rc1 As CadObj, rc2 As CadObj
  MapPro1.Cad.Clear
  Pt1X = -100
  Pt1Y = 30
  Pt2X = -105
  Pt2Y = 34
  Pt3X = -102
  Pt3Y = 36
  Set Bz = MapPro1.Cad.Bezier(Pt1X, Pt1Y, Pt2X, Pt2Y, Pt3X, Pt3Y)
  Bz.Caption = "Bezier[1]"
  ' Show the Bezier control lines
  Set Ln1 = MapPro1.Cad.mLine(Pt1X, Pt1Y, Pt3X, Pt3Y)
  Ln1.Caption = "Line[2]"
  Set Ln2 = MapPro1.Cad.mLine(Pt2X, Pt2Y, Pt3X, Pt3Y)
  Ln2.Caption = "Line[3]"
  Ln1.Pen.Style = 3
  Ln1.Pen.Color = vbBlue
  Ln2.Pen.Style = 3
  Ln2.Pen.Color = vbBlue
  ' Get and then use the handles to the built-in markers.
      s1 = MapPro1.Cad.GetMarker(2)
      s2 = MapPro1.Cad.GetMarker(4)
      s3 = MapPro1.Cad.GetMarker(7)
  Set Pt1 = MapPro1.Cad.Marker(Pt1X, Pt1Y, s1)
  Set Pt2 = MapPro1.Cad.Marker(Pt2X, Pt2Y, s2)
  Set Pt3 = MapPro1.Cad.Marker(Pt3X, Pt3Y, s3)
  Pt1.Caption = "point[4]"
  Pt2.Caption = "point[5]"
  Pt3.Caption = "point[6]"
  Bz.Pen.Width = 4
  Bz.Pen.Color = vbGreen
  ' create a circle and a triangle
  Set rc1 = MapPro1.Cad.mCircle(-100, 30, -80, 38, 2)
  rc1.Caption = "Circle[7]"
  Set rc2 = MapPro1.Cad.RegularPolygon(-100, 30, -90, 36, 3)
  rc2.Caption = "Triangle[8]"
  ' Send the last object ToBack, i.e. below the other objects
  With MapPro1.Cad
    .Objects(.Count - 1).Selected = True
```

```
    .SendToBack
  End With
  MapPro1.Refresh
End Sub
```

.CAD.Symbol(X1,Y1,X2,Y2:Double; Hnd:LongInteger):CadObj — Method

Draws a symbol within the specified bounding rectangle, using the built-in symbol metafile specified through the Hnd parameter, i.e., using its handle. The metafile for the selected build-in symbol is automatically scaled to fit within the bounding rectangle, and is then played on the screen.

VB Example

```
Private Sub Command60_Click()
Dim hndl As Long, SymO As CadObj
  ' Get a built-in meta symbol handle
  hndl = MapPro1.Cad.GetSymbol(2)
  ' USe handle to create object
  Set SymO = MapPro1.Cad.Symbol(-100, 30, -120, 41, hndl)
  ' Set some symbol attributes
  SymO.Pen.Width = 4
  SymO.Pen.Color = vbBlue
  MapPro1.Refresh
End Sub
```

.CAD.Text(X,Y: Double; s:String) — Method

Creates a text object with the caption specified by **S** and at the specified coordinates. Note that the precise placement of the text is also affected by the ***.Font.Align*** value.

VB Example

```
Private Sub Command31_Click()
Dim TTx As CadObj
  ' Create a Text Object
  Set TTx = MapPro1.Cad.Text(-100, 42, "Sample Text")
  TTx.Font.Height = 42
  TTx.Font.BackColor = vbRed
  TTx.Font.Color = vbBlue
  TTx.Font.Style = 3
  MapPro1.Refresh
  ' Change the object caption (mod will not be seen till nect
refresh)
  TTx.Caption = "New one"
End Sub
```

.CAD.TextBubble(X,Y:Double, S:String) — Method

Creates a TextBubble at the specified Lon,Lat location containing the specified string.

VB Example

```
Private Sub Command49_Click()
Dim Txb As CadObj
  ' Create Text Bubble
  Set Txb = MapPro1.Cad.TextBubble(-80, 32, "Sample Text Bubble")
  MapPro1.Refresh
End Sub
```

.CAD.TextFloat(X1,Y1,Size:Float; Caption:String):CadObj — Method

Creates a CAD text object using the current font attributes. The Size of the text is tied to the current scale value and it resizes automatically as the user zooms in/out.

X1,Y1 - are the lon/Lat coordinates of the reference point for the text object.

Size - is the font size in degrees. Note that by definitions this is a dynamically resizable object, so this size is tied to the current viewport scale. Also take a look at the CAD Helper function *ScaleToSize* which was specifically developed to help the developer with the use of TextFloat.

Delphi Example

```
procedure TForm1.Button17Click(Sender: TObject);
var tmk1, tmk2:array [1..10] of CadObj;
    i:integer;
    x:double;
begin
  // Set 2 TextFloat and 2 Text objects
  For i := 1 To 2 do
  begin
    // Get the value to use for 12 pixel height at 5 mile scale
    x:=mappro1.cad.scaletosize(5,12);
    tmk1[i] := MapPro1.Cad.Textfloat(-110 + (i * 0.02),
            30 + (i * 0.02), x, 'Item: ' + inttostr(i));
    tmk1[i].Font.color := clblue;
    tmk2[i] := MapPro1.Cad.Text(-110 - (i * 0.04),
            30 + (i * 0.04), 'Item: ' + inttostr(i));
  end;
  mappro1.GotoPoint(-110,30);
  mappro1.miles:5;
end;
```

.CAD.Toolbar — Interface

Controls the CAD Toolbar, which gives the user access to all the CAD capabilities through a visual interface, rather than having to implement them programmatically. See the CAD.Toolbar section for a detailed description of the Toolbar's functionality.

Toolbar.mode
Sets the mode of the CAD Toolbar. The following three enumerated modes are available.

TbFix – Anchors the CAD toolbar at the top of the application form.
TbFloat – Creates a floating instance of the CAD toolbar
TbHide – Hides the CAD Toolbar

ToolBar.SetPos - Reserved

.CAD.Ungroup() Method

Breaks a grouped CAD object down to the components. (Also see the ***.CAD.Group*** method)

VB Example
```
Private Sub Command65_Click()
  ' Group selected objects
  MapPro1.Cad.Group
  ' Update to show grouped objecs.  Mode to show handles.
  MapPro1.MapMode = MdCad
  MapPro1.Cad.ObjectType = 0
  MapPro1.Refresh
End Sub
```

.CAD.Visible Method

Controls the visibility of the whole CAD layer. A convenient way to turn all objects to visible/invisible.

VB Example
```
Private Sub Command61_Click()
  ' Toggle the CAD layer visibility
  MapPro1.Cad.Visible = Not (MapPro1.Cad.Visible)
  MapPro1.Refresh
End Sub
```

CAD Object Properties

All CAD objects defined in MapPro have a common set of properties as defined below.

.objects().Brush Property

Holds the brush that is used to paint (fill) the interiors of defined objects (polygons, circles, etc.) Default brush is Solid, White, Opaque.

.brush.BackColor Defines the brush background color

.brush.Color Defines the brush color

.brush.Mode Two brush modes are available. 1 – Transparent and 2- Opaque.
Note that the mode of the **pen**, which controls raster operations, is used to define the raster operation of the brush, since the pen object is used for the brush as well.

.brush.Style Constants specifying the brush style (note that the actual line style is controlled by the pen attribute).

0 – Solid
1 – Clear
2 – Horizontal
3 – Vertical
4 – Left Dash
5 – Right Dash
6 – Cross
7 – Diagonal Cross

VB Example

```
Private Sub Command35_Click()
Dim Circ As CadObj
   Set Circ = MapPro1.Cad.mCircle(-80, 40, -60, 40, 3.5)
  'Set some properties
  Circ.Brush.Color = vbRed
  Circ.Brush.BackColor = vbYellow
  Circ.Brush.Style = 3  'constants only
  Circ.Brush.Mode = 1 ' Transparent 2 is solid
  Circ.Pen.Color = vbGreen
  Circ.Pen.Style = vbSolid
  'Refresh the map
  MapPro1.Refresh
End Sub
```

.objects().Caption Property

Holds the caption text associated with each object. The attributes for the text in the caption are set through the *Object.Font* interface. Note that for Line objects, the caption is automatically aligned parallel with the line in the From-To direction. Any *Font.Angle* set for line objects is added to the automatically calculated angle used for the above arallel placement.

VB Example

```
Private Sub Command31_Click()
Dim TTx As CadObj
  ' Create a Text Object
  Set TTx = MapPro1.Cad.Text(-100, 42, "Sample Text")
  TTx.Font.Height = 42
  TTx.Font.Mode = -1
```

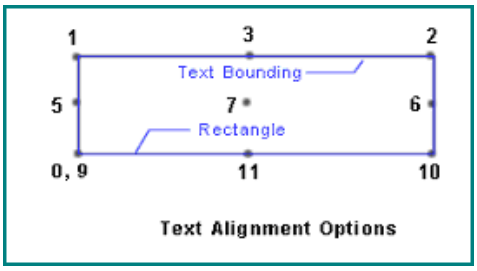
```
  TTx.Font.BackColor = vbRed
  TTx.Font.Color = vbBlue
  TTx.Font.Style = Val(Text18.Text)
  MapPro1.Refresh
  ' Change the object caption (mod will not be seen till nect
refresh)
  Tx.Caption = "New Caption for Text Object"
End Sub
```

.objects().Font **Property**

Controls the attributes of the text object, as well as of the attributes of the ***.Caption*** of every other CAD object.

.font.Align: Integer
Controls the alignment of the text within the text bounding rectangle. Note that the reference point of any text string is the bottom left corner of the string.

The default vertical aligment is zero, which sets the top of the text bounding rectangle aligned with the reference point. There are basically 3 primary top alignment options, 1=Left, 2=Right, 3-Center. Those options *"**OR**ed with 4*" also provide *"middle"* vertical alignment, while *"**OR**ed with 8"* provide *"bottom"* vertical alignment.



.font.Angle:Integer
Sets the angle the text is to be rotated through, in degrees. Positive direction is counter-clockwise.

.font.BackColor:Integer
Sets the background color of the text. Note that this has any bearing only when ***.font.Mode*** is set to opaque or when the ***.font.Style*** is set to Outline.

.font.Color:Integer
Sets the foreground Font Color.

.font.Height:Integer
Sets the Font height pixels.

.font.Mode:Integer
Two font modes are available. Mode=1 is transparent and Mode=2 is opaque.

.font.Name:String

.font.Size:Integer

Sets the font size in points.

.font.Style:Integer

Three basic font styles are available, which can be combined to achieve a compounded affect.

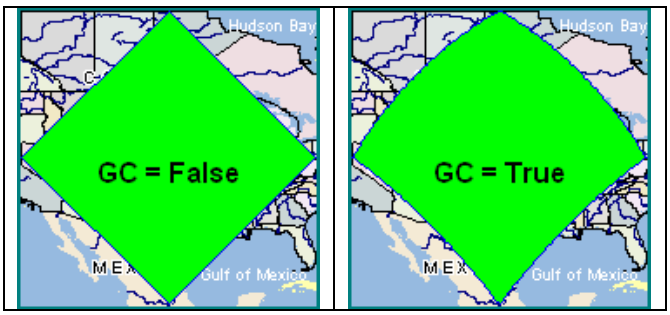
Style #	Font Style
1	Bold
2	Italic
8	Outline

Setting the style to 1+2 = 3, will result in a Bold, Italic style.

VB Example
```
Private Sub Command34_Click()
Dim RegPol As CadObj
  ' Making sure we have a clean slate
  MapPro1.Cad.Clear
  MapPro1.Refresh
  ' Draw Regular Polygon
  Set RegPol = MapPro1.Cad.RegularPolygon(-100, 40, -80, 40, 4)
  RegPol.Font.Style = 8
  RegPol.Font.Align = 3
  RegPol.Font.Color = vbYellow
  RegPol.Font.BackColor = vbRed
  UnFlg = Not (UnFlg)
  If UnFlg = True Then
    RegPol.Caption = "GC = True"
    RegPol.GreatCircle = True
  Else
    RegPol.Caption = "GC = False"
    RegPol.GreatCircle = False
  End If
End Sub
```

.objects().GreatCircle:Boolean **Property**

Controls whether an object is drawn using a straight line, or a Great Circle paradigm.



.objects().MoveAbs(X,Y: Double) Property

Move the object to a new absolute Lon, Lat position. The anchor/reference point is moed to the new location. For example, for Lines, Polylines, etc. the first point of the object is used, whereas for Circles, Polygons, etc. the center point is used.

VB Example

```
Private Sub Command45_Click()
  ' Move the first object to a new location
  MapPro1.Cad.Objects(1).MoveAbs -118, 22
  MapPro1.Refresh
End Sub
```

.objects().MoveRel (X,Y: Double) Property

Moves the object to a new location offset by X,Y degrees from its current location. The first point of the object is used as the reference point for the move.

VB Example

```
Private Sub Command45_Click()
  ' Move the first object to a new location 5 degrees
  ‘ west 2 degrees north from its current location.
  MapPro1.Cad.Objects(1).MoveRel -5, 2
  MapPro1.Refresh
End Sub
```

.objects().ObjectType:Integer Property

Identifies the type of CAD object. The following object types are available in this CAD interface.

#	Object Type
1	Line
2	Rectangle
3	Ellipse
4	Polyline
5	Polygon
6	Marker
7	Text
8	Circle
9	Regular Polygon
10	Free Hand (*)
11	Arrow
12	Bezier
13	Symbol
14	TextBubble
15	Grouped Object
16	MetaObj

(*) This is the ObjectType for the Free Hand object in the GUI. Once the object is created, it is internally converted to a polyline, so an object that was created as a free hand object and the saved, will be retrieved as a polyline object.

.objects().Pen **Property**

Sets the properties for the pen that will be used to draw the object.

.pen.BackColor
The pen background color, i.e., the color seen in the spaces, when something like a dashed-line style is used.

.pen.Color
The main (foreground) pen color.

.pen.Mode
The pen mode controls whether the pen will be opaque (=1) or transparent (=2).

.pen.ROP
The pen raster operation that defines how it will interact with the colors of the map layer.

The Raster Operations vvailable are:

VB Constant	Value	Description
vbBlackness	1	Black
vbNotMergePen	2	Not Merge pen
vbMaskNotPen	3	Mask Not pen
vbNotCopyPen	4	Not Copy pen
vbMaskPenNot	5	Mask pen Not
vbInvert	6	Invert
vbXorPen	7	Xor pen
vbNotMaskPen	8	Not Mask pen
vbMaskPen	9	Mask pen
vbNotXorPen	10	Not Xor pen
vbNop	11	No operation; output remains unchanged
vbMergeNotPen	12	Merge Not pen
vbCopyPen	13	Copy pen
vbMergePenNot	14	Merge pen Not
vbMergePen	15	Merge pen
vbWhiteness	16	White

.pen.Style
Set the style of the pen used to draw the lines of fill patterns. Note that this style

0	Solid
1	Dash
2	Dot
3	Dash-dot
4	Dash-dot-dot
5	Invisible
6	Inside solid

.pen.width

The width of the pen in pixels. Please note that the windows GDI supports line styles other than solids, ONLY for pen thicknesses of one pixel!

VB Example

```
Private Sub Command35_Click()
Dim Circ As CadObj
   Set Circ = MapPro1.Cad.mCircle(-80, 40, -60, 40, 3.5)
  Circ.Brush.Color = vbRed
  Circ.Brush.BackColor = vbYellow
  Circ.Brush.Style = 3  'constants only
  Circ.Brush.Mode = 1 ' Transparent 2 is solid
  Circ.Pen.Color = vbGreen
  Circ.Pen.Style = vbSolid
  Circ.Pen.Width = 4
  'Refresh the map
  MapPro1.Refresh
End Sub
```

objects().Selected:Boolean — Property

A property that indicates if the object is currently selected. Certain methods of the CAD interface (like, Delete, BringToFront, etc.) operate ONLY on selected CAD objects.

.objects().Tag:Integer — Property

This property is reserved for future use. It could, however, be used by the user to relate the current CAD object to some other external file, etc.

.objects().Visible:Boolean — Property

Sets the visibility of the object to True or False.

VB Example

```
Private Sub Command53_Click()
```

```
Dim Bz As CadObj, Pt1 As CadObj, Pt2 As CadObj, Pt3 As CadObj
Dim Ln1 As CadObj, Ln2 As CadObj, rc1 As CadObj, rc2 As CadObj
  MapPro1.Cad.Clear
  ' Set the coordinates for the three points
  Pt1X = -100
  Pt1Y = 30
  Pt2X = -105
  Pt2Y = 34
  Pt3X = -102
  Pt3Y = 36
  Set Bz = MapPro1.Cad.Bezier(Pt1X, Pt1Y, Pt2X, Pt2Y, Pt3X, Pt3Y)
  Bz.Caption = "Bezier[1]"
  ' Show the Bezier control lines
  Set Ln1 = MapPro1.Cad.mLine(Pt1X, Pt1Y, Pt3X, Pt3Y)
  Ln1.Caption = "Line[2]"
  Set Ln2 = MapPro1.Cad.mLine(Pt2X, Pt2Y, Pt3X, Pt3Y)
  Ln2.Caption = "Line[3]"
  Ln1.Pen.Style = 3
  Ln1.Pen.Color = vbBlue
  Ln2.Pen.Style = 3
  Ln2.Pen.Color = vbBlue
  s1 = MapPro1.Cad.GetMarker(2)
  s2 = MapPro1.Cad.GetMarker(4)
  s3 = MapPro1.Cad.GetMarker(7)
  Set Pt1 = MapPro1.Cad.Marker(Pt1X, Pt1Y, s1)
  Set Pt2 = MapPro1.Cad.Marker(Pt2X, Pt2Y, s2)
  Set Pt3 = MapPro1.Cad.Marker(Pt3X, Pt3Y, s3)
  Pt1.Caption = "point[4]"
  Pt2.Caption = "point[5]"
  Pt3.Caption = "point[6]"
  ' Set some attributes
  Bz.Pen.Width = 4
  Bz.Pen.Color = vbGreen
  ' create a circle and a triangle
  Set rc1 = MapPro1.Cad.mCircle(-100, 30, -80, 38, 2)
  rc1.Caption = "Circle[7]"
  Set rc2 = MapPro1.Cad.RegularPolygon(-100, 30, -90, 36, 3)
  rc2.Caption = "Triangle[8]"
  ' Set the first and last object to invisible
  With MapPro1.Cad
      .Objects(0).Visible = False
      .Objects(.Count - 1).Visible = False
    .SendToBack
  End With
  MapPro1.Refresh
End Sub
```

Visual CAD Toolbar Interface

The CAD capabilities in MapPro are also accessible though the control's built-in CAD toolbar, which is invoked through the .CAD.Toolbar interface. Once the Toolbar is instantiated, the following capabilities are available to the user:

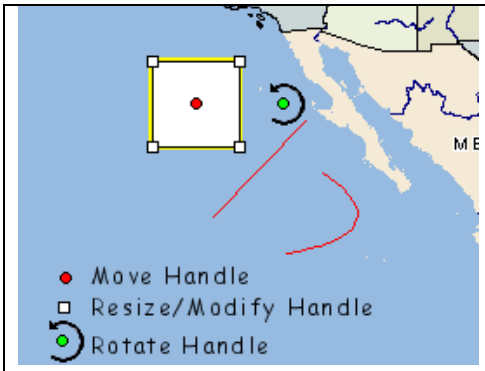
Note that the user may click on the blue caption area of the toolbar and drag it to a new location. In addition to that, the user may Right-Click on the two vertical handles (left side of the toolbar) and select the option to Anchor, UnAnchor and Close.

This is the Selection action button. Clicking on it toggles the control from an Object-Selection mode, when it's depressed (the mouse pointer changes, as well), to a normal mdZoom mode, when it is not depressed. While in the select mode, the user can select objects for subsequent editing action, either by clicking and dragging the mouse pointer, i.e., enclosing the objects in the dynamically drawn selection rectangle, or by clicking inside or on the object itself. When an object is selected, it's highlighted in Yellow.

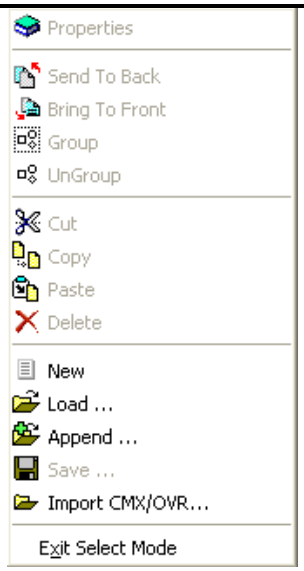
Holding down the Shift button, while selecting individual objects, adds them to the already selected list.

When an object is selected, it's control points are made visible. In addition to that, a red control point is shown (for moving the object) and a rotate control point is shown for rotation the object.

The Edit/Select mode stays active until the user clicks on the button once again. Ckiclinng on the Down-Arrow portion of the button invokes the pop-up menu (see later on).



When an object is selected, it displays three types of handles.
(a) The solid red handle is the ***Control Handle*** which allows the user to move an object by clicking on it an dragging the mouse cursor. (b) the square white handle with black border is simply a ***Vertex*** that can be used to resize or modify the shape of the object (where permitted). (c) the ***Rotation*** handle that permits certain objects to be rotated about their control handle.

<table>
<tr><td></td><td>
<table>
<tr><td> 
</td><td>If an object (or multiple objects) are selected and the user clicks the Right mouse button, then a pop-up menu appears giving the user a number of editing options, as shown to the left. Most of the commands available are self-explanatory, or are explained in the main portion of the CAD documentation.

• Properties opens up the dialog the allows the user to change the attributes of the selected object(s).
• Load… and Save… permit the user to Load a CAD file (clearing any current objects), or save the current CAD objects to disk.
• Append allows the user to Load a CAD file and append its objects to those currently defined.
• There is also an extra option that allows the user to load old-style CMX/OVR files, which are transparently converted to the new CAD format.</td></tr>
</table>
</td></tr>
<tr><td> </td><td>This is button will also invoke the pop-up menu, described earlier, and doesn not require that any objects are selected.</td></tr>
<tr><td> </td><td>This is a dual action button. Clicking on the brush portion, sets the default brush/pen color to that displayed at the bottom portion of the button. If any objects are selected, then the pen/brush color of those objects is set to that color as well.

Clicking on the down-arrow portion of the button, opens up a color selection dialog that allows the user to set the pen/brush color that will be used for any subsequent Objects.

Note that if any objects are selected, then the pen/brush color of those objects is set to that color as well.

It should also be pointed out that the pen/brush color for selected objects, may also be modified by opening the Attributes dialog (see description later on)</td></tr>
<tr><td> </td><td>This is a dual action button. Clicking on the fill-bucket portion, sets the default fill color to that displayed at the bottom portion of the button. If any objects are selected, then the fill color of those objects is set to that color as well.

Clicking on the down-arrow portion of the button, opens up a color selection dialog that allows the user to set the fill color that will be used for any subsequent Objects.

Note that if any objects are selected, then the fill color of those objects is set to that color as well.

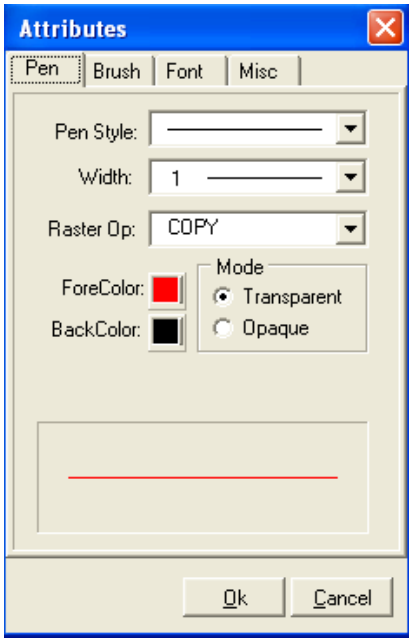
It should also be pointed out that the fill color for selected objects, may also be modified by opening the Attributes dialog (see description later on)</td></tr>
<tr><td> </td><td>This is a dual action button. Clicking on the brush portion, sets the default font color</td></tr>
</table>

	to that displayed at the bottom portion of the button. If any objects are selected, then the font color of those objects is set to that color as well. Clicking on the down-arrow portion of the button, opens up a color selection dialog that allows the user to set the font color that will be used for any subsequent Objects. Note that if any objects are selected, then the font color of those objects is set to that color as well. It should also be pointed out that the font color for selected objects, may also be modified by opening the Attributes dialog (see description later on)
	Opens up the attributes dialog that allows the user to set the default pen, fill and font attributes that become the default for any subsequently drawn object. If any objects are selected, then the attributes of those objects are also modified as set by the user. See the section on CAD Attributes for a detailed description of these dialogs.
	Draw a straight line segment. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets the starting point of the line segment. Moving the cursor and left-clicking again, sets the end point of the segment, and the start point of the next segment, i.e., the user may create connected straight line segments by sequentially moving the pointer and clicking the left mouse button. The drawing mode is cancelled by clicking the right mouse button, and the screen is updated to reflect the currently drawn object.
	Draw a rectangle. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets one of the corners of the rectangle, the anchor corner. Dragging the mouse pointer dynamically resizes the rectangle until the user clicks the left mouse button again, at which time the rectangle object is created, the screen is updated to reflect the newly created object, and the "object-creation" mode is exited.
	Draw an ellipse. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets one of the corners of the bounding rectangle of the ellipse, the anchor corner. Dragging the mouse pointer dynamically resizes the bounding rectangle, and the ellipse, until the user clicks the left mouse button again, at which time the ellipse object is created, the screen is updated to reflect the newly created object, and the "object-creation" mode is exited.
	Draw a circle. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets the center of the circle. Dragging the mouse pointer dynamically resizes the circle until the user clicks the left mouse button again, at which time the circle object is created, the screen is updated to reflect the newly created object, and the "object-creation" mode is exited.
	Draw a regular polygon. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets the

	centroid of the regular polygon. Dragging the mouse pointer dynamically resizes the regular polygon until the user clicks the left mouse button again, at which time the object is created, the screen is updated to reflect the newly created object, and the "object-creation" mode is exited. Note that the number of sides of the polygon is set through the CAD Attributes dialog, described later on.
	Draw a PolyLine Object. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets the starting point of the object. Moving the cursor and left-clicking again, sets the end point of the current segment of the polyline, and the start point of the next segment. The drawing mode is cancelled by clicking the right mouse button, and the screen is updated to reflect the currently drawn object.
	Draw a Polygon Object. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets the starting point (first vertex) of the polygon. Moving the cursor and left-clicking again, sets subsequent vertices. The drawing mode is cancelled by clicking the right mouse button, and the screen is updated to reflect the currently drawn object.
	Draw an Arrow Object. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets the point of the arrow. Moving the cursor and left-clicking again, sets the tail (end) point of the arrow, the drawing mode is cancelled, and the screen is updated to reflect the currently drawn object.
	Draw a Freehand Object. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets the starting point of the object. Moving the cursor, while keeping the left mouse button pressed, dynamically creates a trace of the mouse movement. The drawing mode is cancelled by releasing the mouse button, and the screen is updated to reflect the currently drawn object.
	Draw a Floating Size Text Object. The text is drawn at the selected size at the current map scale, and is then proportionally scaled as the map scale changes.
	Draw a Text Bubble Object. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets the pointer of the text bubble, opens up an edit box and allows the user to type in the desired text. When ENTER is pressed, the drawing mode is cancelled and the screen is updated to reflect the currently drawn object.
	Draw a Text Object. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets the reference point for the text object, opens up an edit box and allows the user to type in the desired text. When ENTER is pressed, the drawing mode is cancelled and the screen is updated to reflect the currently drawn text object. Note that the size of the text remans the same regardless of the current map scale.
	Draw a Bezier Object. The cursor changes into a small cross hair to indicate that the control is in the "object-drawing" mode. Clicking the left mouse button sets the starting point of the object. Clicking a second time sets the and point of the bezier

	object. Moving the cursor, dynamically modifies the bezier by varying its third control point. The bezier object is set by pressing the left mouse button for a third time. The drawing mode is cancelled and the screen is updated to reflect the currently drawn object.
	This is a dual action button. Clicking on the object portion, select the current Windows metafile (has to have been selected by the user) and sets the object creation mode. The cursor changes to a small cross-hair. Clicking the left mouse button sets one of the corners of the metafile bounding polygon. Dragging the mouse pointer dynamically resizes the bounding rectangle until the user clicks the left mouse button again, at which time the metafile object is created. Clicking on the down-arrow portion of the button, opens up a stock windows dialog that permits the user to navigate and select a metafile to be used as the default, until a subsequent, new selection.
	This is a dual action button. Clicking on the icon portion, selects the current built-in metafile object (also reflected on the button's surface). The cursor changes to a small cross-hair. Clicking the left mouse button sets the current metafile symbol at the user-selected location, and leaves it in a resize mode, i.e., dragging the cursor dynamically resizes the object. Until the left mouse button is pressed. Clicking on the down-arrow portion of the button, opens up a selection list with built-in simple metafile objects, and allows the user to select one of them for subsequent placement on the map surface. Note that unlike the user-specified metafile objects that cannot have their attributes altered, the pen and brush color and line styles of this group of objects may be modified by the user, after the object is placed on the map surface. If the symbol library file is not found, then the button displays **"No Lib"**, instead.
	This is a dual action button. Clicking on the icon portion, select the current Marker and sets the object creation mode. The cursor changes to a small cross-hair. Clicking the left mouse button sets the current marker at the user-selected location. Clicking on the down-arrow portion of the button, opens up a dialog that permits the user select a new marker type. The new marker is reflected on the icon portion of the dual action button.
	Anchors the toolbar to the bottom of the control frame.
	Closes the Toolbar.

If the CAD toolbar is visible, clicking on the Attributes button opens up the attributes dialog, which permits the user to set the default attributes for any CAD objects that are subsequently created. If any CAD objects are currently selected, then the attributes of the selected objects are set to those selected by the user. The Attribute dialog is described below.



CAD Attributes Dialog – Pen Tab

Pen Style – The user may select one of the available pen styles (solid, dotted, dash, etc.)

Width – The user may select the pen width, in pixels. Note that Windows limitations only permit lines of *thickness = 1 pixel* to have any pan style other than solid.

Raster Op – The user may select one of four main Windows raster operations: COPY, MERGER, MASK, XOR.

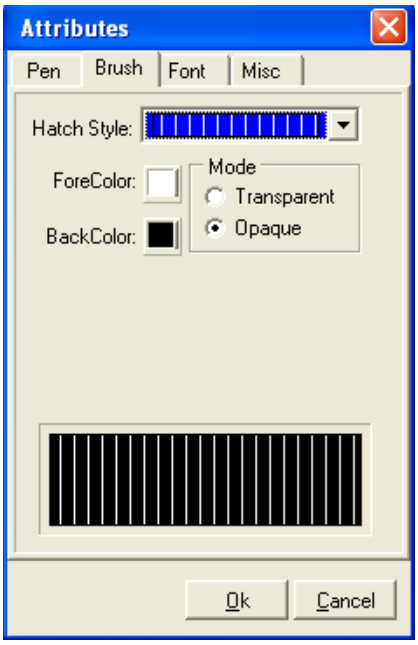
ForeColor – Clicking on the color button, a color selection dialog opens up and lest the user select the pen foreground color.

BackColor – Clicking on the color button, a color selection dialog opens up and lest the user select the pen background color, i.e., the color visible underneath when a line style other than solid is used.

Mode – The user may select either a transparent or an opaque pen mode.

The currently selected pen attributes are reflected in the bottom portion of the dialog.

CAD Attributes Dialog – Brush Tab



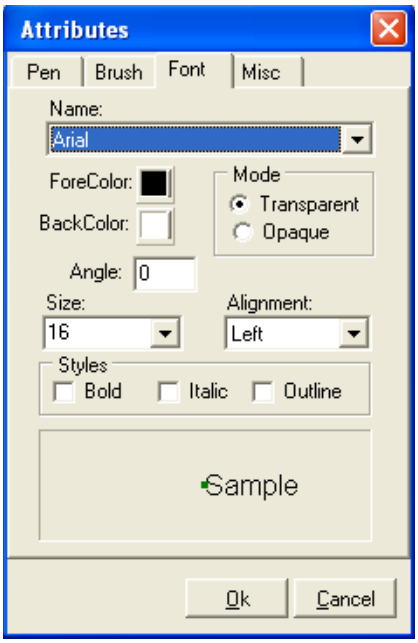
Hatch Style – The user may select one of the available hatch styles (vertical, horizontal, slash, solid, etc.)

ForeColor – Clicking on the color button, a color selection dialog opens up and lest the user select the brush foreground color.

BackColor – Clicking on the color button, a color selection dialog opens up and lest the user select the brush background color.
Mode – The user may select either a transparent or an opaque pen mode.

The currently selected pen attributes are reflected in the bottom portion of the dialog.

CAD Attributes Dialog – Font Tab



Name – The user may select one of the available installed Windows Fonts.

ForeColor – Clicking on the color button, a color selection dialog opens up and lest the user select the brush foreground color.

BackColor – Clicking on the color button, a color selection dialog opens up and lest the user select the brush background colo.

Mode – The user may select either a transparent or an opaque pen mode (applies to the background color).

Angle – The user may specify an angle that the Text is to be rotated through, in degrees (positive is counter clockwise).

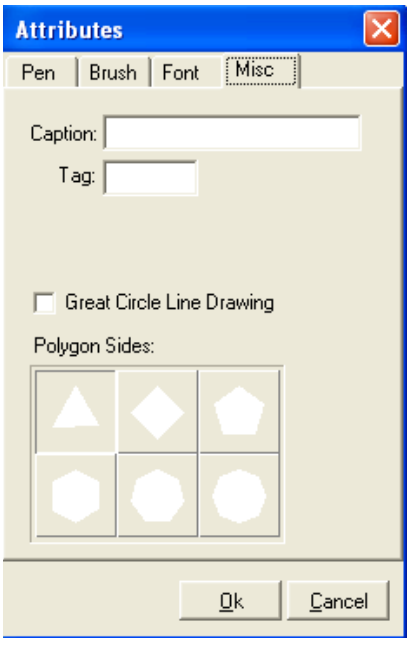
Size – The font size in points

Alignment – The user may select one of the following text alignments (relative to the text reference point specified): TopLeft, TopRight, TopCenter, Left, Right, Center, BotLeft, BotRight, BotCenter.

Styles – The available styles may be combined as desired by the user.

The currently selected pen attributes are reflected in the bottom portion of the dialog.

CAD Attributes Dialog – Misc Tab



Caption – The user may enter the default caption to appear for any subsequently created object.

Polygon Sides – Clicking on one of the predefined buttons sets the number of Regular Polygon sides, or the user may enter the number of desired sides.

Tag – An extra identifier that can be used by the user.

Great Circle Line Drawing – If selected, affected objects are drawn with Great Circle Lines, instead of straight lines.

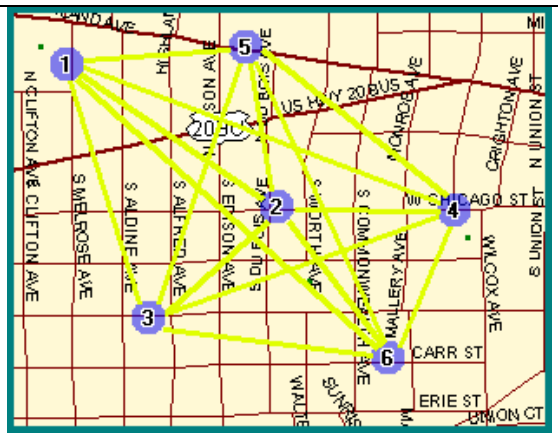
CAD Attributes Cross-Reference Table

Although all CAD objects seem to include all available attributes (for object consistency) not all attributes are accessible/usable for all objects, as shown in this summary table.

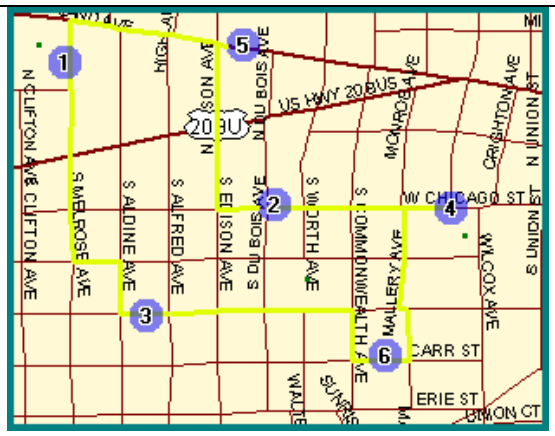
	Brush	Caption	Font	Great Circle	MoveAbs	MoveRel	ObjectType	Pen	Selected	Tag	Visible
Line		x	x	x	x	x	x	x	x	x	x
Rectangle	x	x	x	x	x	x	x	x	x	x	x
Ellipse	x	x	x		x	x	x	x	x	x	x
Polyline		x	x	x	x	x	x	x	x	x	x
Polygon	x	x	x	x	x	x	x	x	x	x	x
Marker	x	x	x	x	x	x	x	x	x	x	x
Text		x	x		x	x	x	x	x	x	x
Circle	x	x	x	x	x	x	x	x	x	x	x
Regular Polygon	x	x	x	x	x	x	x	x	x	x	x
Free Hand		x	x		x	x	x	x	x	x	x
Arrow	x	x	x	x	x	x	x	x	x	x	x
Bezier	x	x	x	x	x	x	x	x	x	x	x
Symbol	x	x	x		x	x	x	x	x	x	x
TextBubble	x	x	x	x	x	x	x		x	x	x
Grouped Object	x	x	x	x	x	x	x	x	x	x	x
MetaObj		x									

.OptiRouter Interface

A whole collection of new capabilities were added to the Map control in the new release, MapPro71. One of the most important additions is the Optimized Route Solution (ORS), many times referred to as the Traveling Salesman Problem (TSP). This is the solution to the problem where a number of points need to be visited, and the user is interested in the optimized set of routes to all those points. In this, first, implementation, the ORS assumes a closed circle path, i.e., a complete closed path is generated between **all** the user-specified points. Although it may seem to be a trivial problem, for a couple of points, one can quickly see that the number of possible solutions that need to be examined, even for 6 pint, can quickly get very high.



Possible Routing Combinations for 6 points



ORS result for the same point

The ORS is implemented as a collection of Interfaces, Properties and Methods as described below. Its capabilities can be selected both programmatically, or by invoking the RouteOptimizer built-in dialog.

.OptiRouter.AddPoint(s:String, x,y:Double) — Method

Adds the specified point to the optimization routing points list. Note that the points currently defined may also be accessed at any time using the StopPoints array.

where: **s** = Any string identifying the specified point
x,y = The x,y (Lat/Lon) coordinates of the point

Points are appended at the end of the list of any points that have already been added. Also see the MovePoint, DeletePoint and Insert point methods of the OptiRouter interface.

VB Example

```
Private Sub Command3_Click()
Dim s1b As Double, s1c As Double, s2b As Double, s2c As Double
Dim s3b As Double, s3c As Double, s4b As Double, s4c As Double
' Do some address look-ups and add the points returned
  MapPro1.GeoFind "231 sutton Street,North Andover,MA, 01845"
  s1a = MapPro1.GeoFindParse(1, MapPro1.Street)
  s1b = Val(MapPro1.GeoFindParse(7, MapPro1.Street))
  s1c = Val(MapPro1.GeoFindParse(8, MapPro1.Street))
```

```
        MapPro1.OptiRouter.AddPoint s1a, s1b, s1c
        List1.AddItem "Just Set: " & s1a & ",  " & s1b & ",  " & s1c
        '---
        MapPro1.GeoFind "100 Main St, N. Andover, MA"
        s2a = MapPro1.GeoFindParse(1, MapPro1.Street)
        s2b = Val(MapPro1.GeoFindParse(7, MapPro1.Street))
        s2c = Val(MapPro1.GeoFindParse(8, MapPro1.Street))
        MapPro1.OptiRouter.AddPoint s2a, s2b, s2c
        List1.AddItem "Just Set: " & s2a & ",  " & s2b & ",  " & s2c
        '---
        MapPro1.GeoFind "245 Summer Street, Boston, MA"
        s3a = MapPro1.GeoFindParse(1, MapPro1.Street)
        s3b = Val(MapPro1.GeoFindParse(7, MapPro1.Street))
        s3c = Val(MapPro1.GeoFindParse(8, MapPro1.Street))
        MapPro1.OptiRouter.AddPoint s3a, s3b, s3c
        List1.AddItem "Just Set: " & s3a & ",  " & s3b & ",  " & s3c
        '---
        MapPro1.GeoFind "207 Market Street, Lowell, MA 01852"
        s4a = MapPro1.GeoFindParse(1, MapPro1.Street)
        s4b = Val(MapPro1.GeoFindParse(7, MapPro1.Street))
        s4c = Val(MapPro1.GeoFindParse(8, MapPro1.Street))
        MapPro1.OptiRouter.AddPoint s4a, s4b, s4c
        List1.AddItem "Just Set: " & s4a & ",  " & s4b & ",  " & s4c
        '---
        'Set some Mark properties (even after the fact)
        MapPro1.OptiRouter.RoutePath.MarkType = 1
        MapPro1.OptiRouter.RoutePath.MarkSize = 20
        MapPro1.OptiRouter.RoutePath.MarkColor = vbRed

        ' Use a delta Offset to make sure Marks are visible (Ref
      point is Mak center)
        Delta = 0.01

        ' Zoom to the extents of the defined points
        MapPro1.ZoomWindow MapPro1.OptiRouter.PointExtent.Xmax +
      Delta, MapPro1.OptiRouter.PointExtent.Ymax + Delta,
      MapPro1.OptiRouter.PointExtent.Xmin - Delta,
      MapPro1.OptiRouter.PointExtent.Ymin - Delta
      End Sub
```

Delphi Example

```
procedure TForm1.Button1Click(Sender: TObject);
var i,k,code:integer; x,y:double;
    temp:shortstring;
    r:tpointrec ;
begin
  MapPro1.OptiRouter.ClearPoints;
  for i:=0 to listbox1.items.count-1 do
  begin
    temp:=listbox1.items[i];
    //---
    k:=pos(',',temp);
    val(copy(temp,1,pred(k)),x,code);
    val(copy(temp,succ(k),length(temp)-k),y,code);
    MapPro1.OptiRouter.AddPoint('Start-'+inttostr(i),x,y);
    r:=MapPro1.OptiRouter.stopPoints[i];
    listbox3.Items.add(r.name+', '+floattostr(r.x)+',
'+floattostr(r.y));
  end;
  label1.caption:=Inttostr(MapPro1.OptiRouter.NumPoints);
end;
```

.OptiRouter.Calculate(n:integer) Method

Performs the route optimization calculation, using the parameters currently specified by the user. The parameter N, specifies whether to return all individual road segments, or consolidate all segments with the same name in the reporting of the calculated route. **N=0** reports **ALL** individual segments, **N=1** consolidates same name segments.

VB Example

```
Private Sub AddPt_Click()
  ' Use the AddPoint to add 4 points to the OptiRouter
  MapPro1.OptiRouter.AddPoint "Poin  1", -88.299318, 42.039389
  MapPro1.OptiRouter.AddPoint "Point 2", -88.306011, 42.036385
  MapPro1.OptiRouter.AddPoint "Point 3", -88.303308, 42.031365
  MapPro1.OptiRouter.AddPoint "Point 4", -88.296486, 42.033983
  ' Zoom to the defined points extents
  With MapPro1.OptiRouter.PointExtent
    MapPro1.ZoomWindow .Xmin, .Ymin, .Xmax, .Ymax
  End With
  ' Save the defined points for later use
  MapPro1.OptiRouter.SavePoints "MyTestOptiPts.pts"
  'Now Calculate the Optimized Route
  MapPro1.OptiRouter.Calculate (0)
End Sub
```

.OptiRouter.Clear() Method

Clears the current Nodes and road segment network that was generated durion a prior .Calculate call. This is different than the .ClearPoints method which clears the routing points.

VB Example

```
Private Sub AddPt_Click()
  ' Clear OptiRouting Nodes/segments
  Mappro1.OptiRouter.Clear
End Sub
```

.OptiRouter.ClearPoints() Method

Clears Only the currently defined Priority Routing points. But leaves all the rest of the parameters currently specified for this module, intact.

VB Example

```
Private Sub AddPt_Click()
  ' Clear OptiRouting Points ONLY
  Mappro1.OptiRouter.ClearPoints
End Sub
```

.OptiRouter.DeletePoint(n:Integer) Property

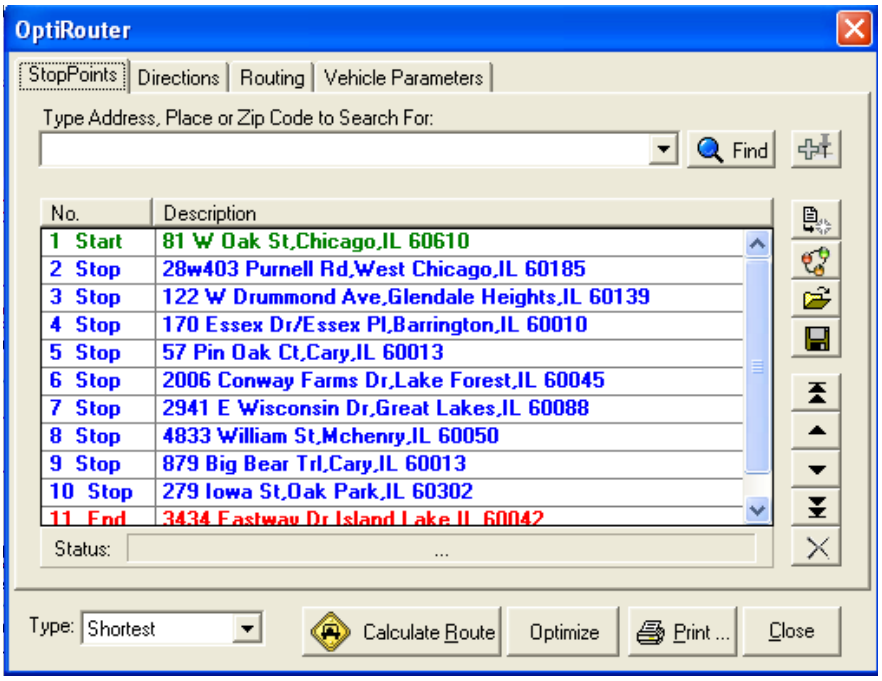
Delete the specified OptiRouter point. Remaining points are moved up a position in the points array. If *N> NumPoints*, then o action is taken.

VB Example

```
Private Sub Command8_Click()
  MapPro1.OptiRouter.DeletePoint (Val(Text4.Text))
  MapPro1.Refresh
End Sub
```

.OptiRouter.ExecOptiRoute() Method

Opens up a dialog to permit the user to specify all the Route Optimization parameters. The first tab of the Optirouter dialog permist the user to search for and add Stop Points to be used in the calculation.

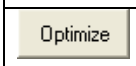
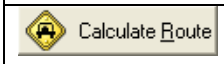
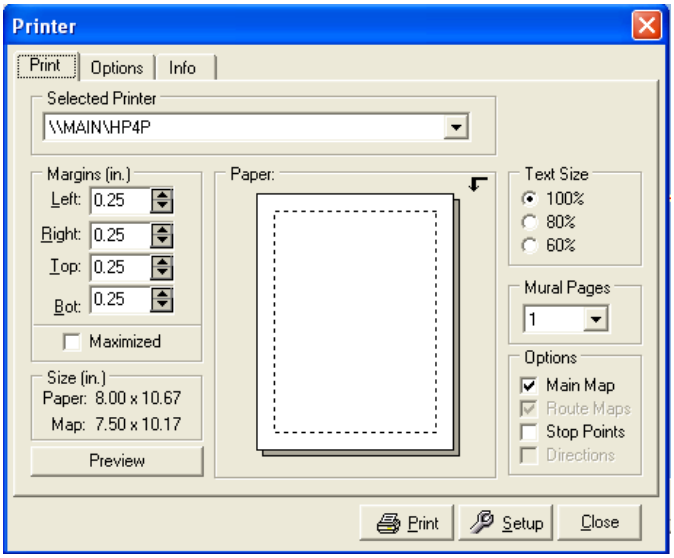
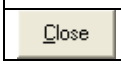


In addition to being able to search for an address to be added in as a stop point, this dialog also allows the user to "bring over" via points that may have been specified for a routing calculation.

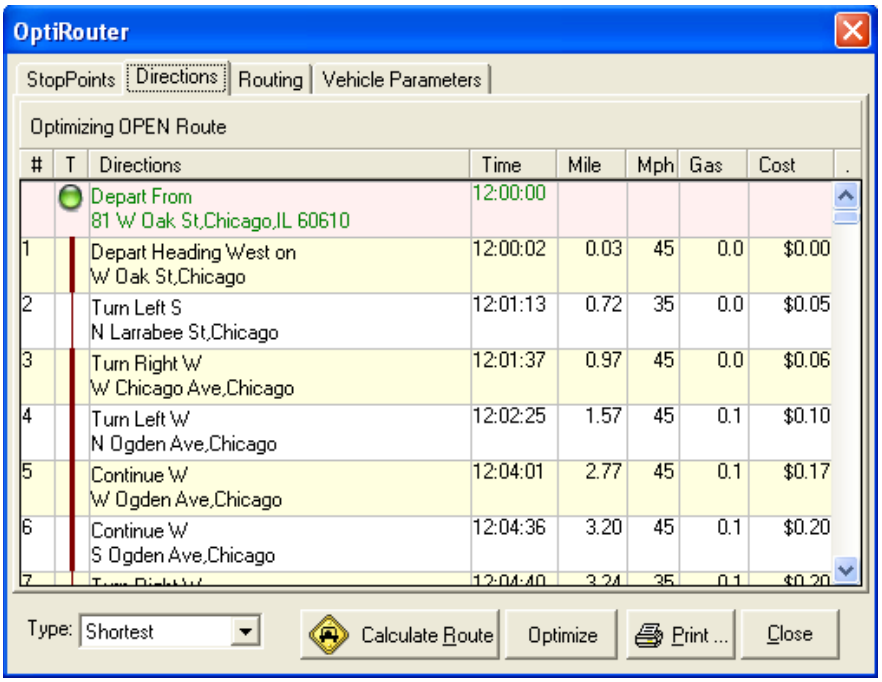
This, in turn, makes it possible for the user to specify Stop Points by Point-and-click. (See introsuction section on Mouse Button functionality).

The functionality of all the buttons available in the StopPoints tab of the Optirouter dialog is explained below.

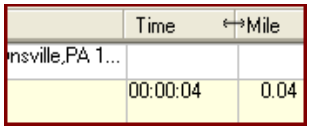
Find	Search for the address typed in the edit box
	Add the "qualified" address (following a search) that appears in the edit box, to the list of Stop Points.
	Clear the Sop Points list. It also clears the markers on the map and the highlighted Optimized Route.
	"Bring Over" any via points that may have been defined and append them to the Stop Points list, immediately above the "End" point.
	Open (Load) a file containing a list of Stop Points.
	Save the current list of Stop Points to a file.

	Make the currently highlighted Stop Point the “Start” point for the Calculation.
	Move the currently highlighted Stop Point UP one position in the list.
	Move the currently highlighted Stop Point DOWN one position in the list.
	Make the currently highlighted Stop Point the “End” point for the Calculation.
	Delete the currently highlighted Stop Point.
	Select the Type of Route to calculate
	Calculate an Optimized route through the current Stop Points (Note that executing this optimize routine, will resequence the stop Points as needed).
	Calculate a route through the current stop points, in the order they appear in the list (not necessarily optimized).
	Open the Print dialog to print the Optimized Route. The same dialog as the one for printing the Via point routing calculation is used. See the Routing sections for a detailed description. 
	Close the OptiRouter dialog.

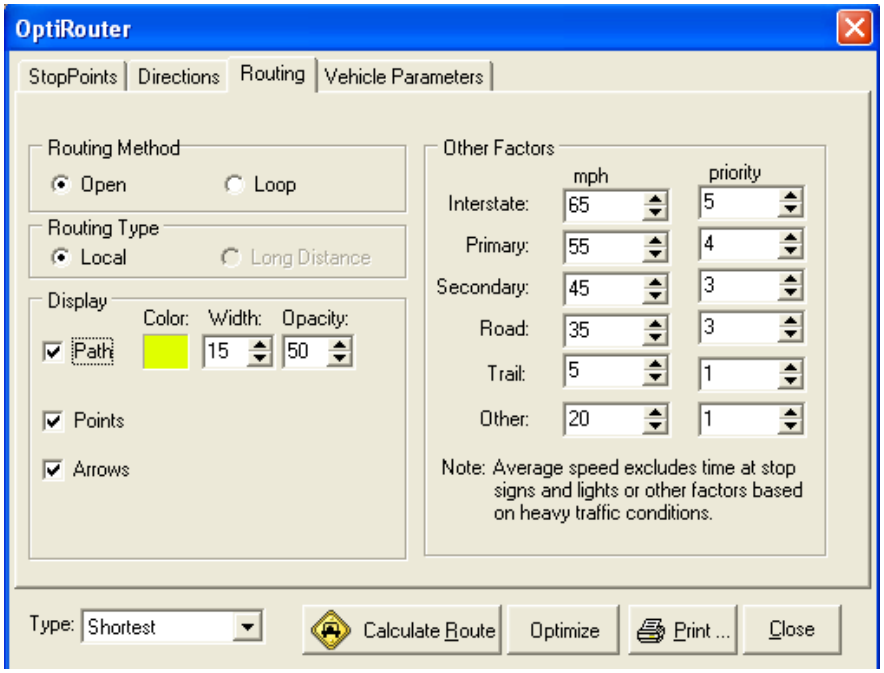
The “Directions” tab of the Optirouter dialog, contains the calculated routing directions, and relative information (distance, cost, etc.), as whown below:



Note that the headers are resizeable. Placing the cursor on one of the vertical dividers and dragging, while holding down the left mouse key, allows the user to resize the column and display more/less information.

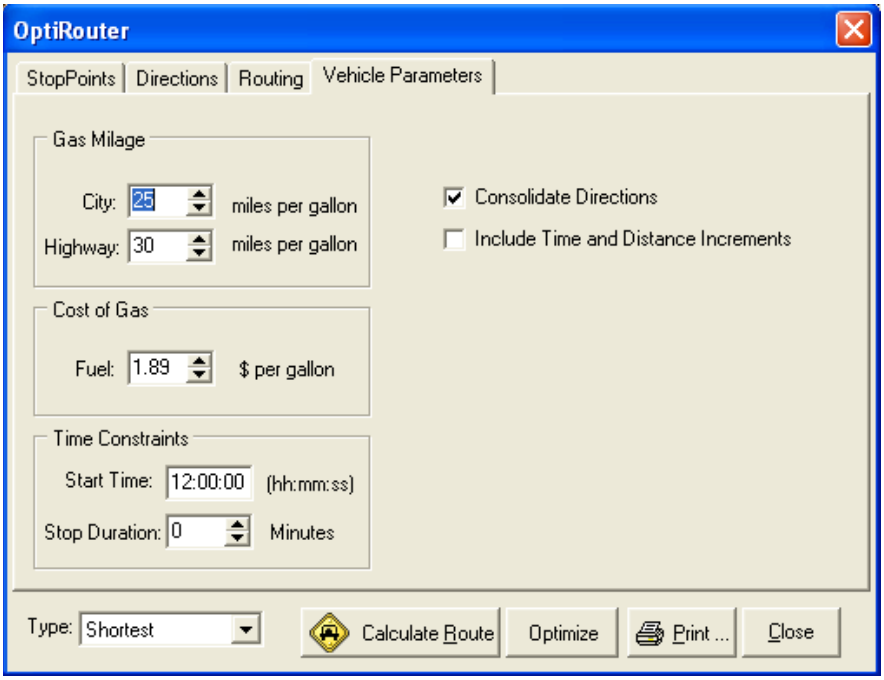


The “Routing” Tab, shown below, allows the user to set a number of the OptiRouter Options, in particular regarding the look of the Stop Points, the calculated route, etc.



Note that the default method “Open”, does an optimization calculation holding the user specified Start and Finish points, in their respective positions, and doing a From/To calculation. If “Loop” is selected, then all points (including Start and Finish) are allowed to be resequenced.

Finally, the vehicle parameters tab, shown below, permits the user to set local and highway MPG consumption, fuel price, etc.



.OptiRouter.GetRouteFirst(n:integer):String — Function

The calculated Routes are placed in a list object and can then be accessed using a GetFirst/GetNext construct. This function takes an argument of zero (reserved for later), as it is assumed that all the required information has already been specified by the user. The function returns the segment # zero, followed by the word Start (tab delimited) to signify the start of the Routing calculations.

VB Example

```
Private Sub GetFirstNext_Click()
  ' Calculate the optimized route using already defined points
  MapPro1.OptiRouter.Calculate (0)
  ' Get the segments and populate the Listbox
  Rseg = MapPro1.OptiRouter.GetRouteFirst(0)
  ' Chek to see if the end of the list object has been reached
  While Rseg <> ""
    List1.AddItem Rseg
    Rseg = MapPro1.OptiRouter.GetRouteNext
  Wend
  ' Close the List object to release
  MapPro1.OptiRouter.GetRouteClose
End Sub
```

.OptiRouter.GetRouteNext():String — Function

The calculated Routes are placed in a list object and can then be accessed using a GetFirst/GetNext construct. This function has to be called **after** a GetFirstRoute call (GetFirstRoute opens the list object), and takes no arguments. The function returns a tab-delimited string containing the following information:

- Sequential Segment #
- Road Classification (CFCC)
- Literal Driving Directions - Continue, Bear, Turn Sharp, and Bearing (SE, NW, etc.). Also includes a Left – Right Modifier. Following the Bearing, the angle is included in square brackets, as well
- Street Name
- Place Name
- Segment Travel time
- Tavel Distance in Miles
- Cumulative Travel time to the end of this segment
- Cumulative Travel Distance to the end of this segment
- Speed during this segment (mi/hr)
- X,Y Start Point coordinates separated by a comma

Since routing between multiple points is performed during these calculations, at the begigning of the routing from **each** point, this function returns 0|Start, just like the GetFirstRoute, to indicate that a new segment between two points is started. When the end of the list object has been reached, the function returns a blank string.

VB Example

```
Private Sub GetFirstNext_Click()
  ' Calculate the optimized route using already defined points
  MapPro1.OptiRouter.Calculate (0)
  ' Get the segments and populate the Listbox
  Rseg = MapPro1.OptiRouter.GetRouteFirst(0)
  ' Chek to see if the end of the list object has been reached
  While Rseg <> ""
    List1.AddItem Rseg
    Rseg = MapPro1.OptiRouter.GetRouteNext
  Wend
  ' Close the List object to release
  MapPro1.OptiRouter.GetRouteClose
End Sub
```

.OptiRouter.GetRouteClose() Function

Closes the OptiRouter list object containing the routing information, and releases all resource, used by the list object, to the system resource pool. It is important to remember that unless this function is called, those resources will not be released.

.OptiRouter.CostPerGallon:Double Property

The cost per gallon, which is used to calculate the cost for the Optimized Route.

.OptiRouter.ExRadius:Double — Property

The search radius about the Optirouter points, for which the network is loaded. The default is 5 miles. Care should be taken not to specify a large radius, since this defines the number of street-level data grids that are loaded, and such grids require significant system resources.

.OptiRouter.GetFirstNode - Reserved — Function

This is reserved for possible future expansion.

.OptiRouter.GetNextNode - Reserved — Function

This is reserved for possible future expansion.

.OptiRouter.StopPoints[n]:PointRec

An indexed array holding all the points that have been specified for the OptiRouter up until this point. (the total number of points is reflected in OptiRouter.NumPoints).

VB Example

```
Private Sub Command2_Click()
  For i = 0 To MapPro1.OptiRouter.NumPoints - 1
    List1.AddItem MapPro1.OptiRouter.StopPoints(i).Name & ": " &
MapPro1.OptiRouter.StopPoints(i).x & ", " &
MapPro1.OptiRouter.StopPoints(i).y
  Next i
End Sub
```

.OptiRouter.InsertPoint (n:Integer; pt:PointRec) — Method

Insert the specified point structure at the Nth position in the currently defined points list.

VB Example

```
Private Sub Command12_Click()
Dim NewPt As TPointRec
  ' define the point
  NewPt.Name = "New point being Inserted"
  NewPt.x = -88.35
  NewPt.y = 42.03
  'Insert a point at position 3
  MapPro1.OptiRouter.InsertPoint 3, NewPt
End Sub
```

.OptiRouter.LoadFrom… Property

Reserved for possible future expansion of the module's capabilities.

.OptiRouter.LoadPoints(s:String) Method

Loads a number of pre-specified point to be used by the OptiRouter, form a text file, replacing ANY points currently defined in memory. The text file format is one line per specified point, with each line containing:

Name, X, Y

VB Example

```
Private Sub Command2_Click()
' Load some points from a point file
MapPro1.OptiRouter.LoadPoints "MyTestOptiPts.pts"
' Now, list the loaded points
  For i = 0 To MapPro1.OptiRouter.NumPoints - 1
    List1.AddItem MapPro1.OptiRouter.StopPoints(i).Name & ": " &
MapPro1.OptiRouter.StopPoints(i).x & ", " &
MapPro1.OptiRouter.StopPoints(i).y
  Next i
End Sub
```

.OptiRouter.MemUsed:Integer Property

The amount of memory (bytes) used to hold the network of points and edges that define the search area around the OptiRouter points.

VB Example

```
Private Sub Command2_Click()
' Get memory used
  s = Mappro1.optirouter.MemUsed
End Sub
```

.OptiRouter.MovePoint(From, To: Integer) Property

Move the point from the specified (From) position to the specified (To) position in the array of the currently defined points.

VB Example

```
Private Sub Command13_Click()
' Load some points from a point file
MapPro1.OptiRouter.LoadPoints "MyTestOptiPts.pts"
' Now, list the loaded points
  For i = 0 To MapPro1.OptiRouter.NumPoints - 1
    List1.AddItem MapPro1.OptiRouter.StopPoints(i).Name & ": " &
MapPro1.OptiRouter.StopPoints(i).x & ", " &
MapPro1.OptiRouter.StopPoints(i).y
```

```
  Next i
' Move Some of the defined points
MapPro1.OptiRouter.MovePoint 1, 3
MapPro1.OptiRouter.MovePoint 2, 4
' And, list the newly ordered points
  For i = 0 To MapPro1.OptiRouter.NumPoints - 1
    List1.AddItem MapPro1.OptiRouter.StopPoints(i).Name & ": " &
MapPro1.OptiRouter.StopPoints(i).x & ", " &
MapPro1.OptiRouter.StopPoints(i).y
  Next i
End Sub
```

OptiRouter.mpgCity:double — Property

Miles per gallon for city driving. Used to calculate the cost for the Optimized Route.

OptiRouter.mpgHwy:double — Property

Miles per gallon for Highway driving. Used to calculate the cost for the Optimized Route.

.OptiRouter.NumPoint(s:String) — Property

The total number of points currently specified in the OptiRouter points array. Note that the points array is a zero-based array, so really NumPoints-1 is the total number of defined points.

VB Example

```
Private Sub Command2_Click()
' Load some points from a point file
MapPro1.OptiRouter.LoadPoints "MyTestOptiPts.pts"
' Now, list the loaded points
  For i = 0 To MapPro1.OptiRouter.NumPoints - 1
    List1.AddItem MapPro1.OptiRouter.StopPoints(i).Name & ": " &
MapPro1.OptiRouter.StopPoints(i).x & ", " &
MapPro1.OptiRouter.StopPoints(i).y
  Next i
End Sub
```

.OptiRouter.PointExtent:TRPoint — Property

Extents of all currently defined OpriRouter Points.

VB Example

```
Private Sub Command101_Click()
  'Zoom Window around OptiRouter Points
  MapPro1.ZoomWindow MapPro1.OptiRouter.PointExtent.Xmin,
```

```
        MapPro1.OptiRouter.PointExtent.Ymin,
        MapPro1.OptiRouter.PointExtent.Xmax,
        MapPro1.OptiRouter.PointExtent.Ymax
End Sub
```

.OptiRouter.Priority:RoadRec Property

Uses a RoadRec record structure to set the priorities for the six main road types. The allowable priority values are 1 (lowest) to 6 (highest), no provision exists to completely exclude a type of road, because in some situations that may be the ONLY road type available. The RoadRec Structure is shown below:

RoadRec - Record Structure
This is a record structure that is used to hold the **speed** and **priority** settings (see OptiRouter Interface properties) for the various road types. There are 6 such road types supported by the system at this point:

.Interstate: LongInt (includes all CFCC A10 classification Roads, i.e., Interstate and limited access highways – some US highways *are* classified as limited access)
.Primary: LongInt (includes all CFCC A20 classification Roads – US Highways that are NOT limited access)
.Secondary: LongInt (includes all CFCC A30 classification Roads)
.Road: LongInt (includes all CFCC A40 classification Roads)
.Trail: LongInt (includes A51, A52 and A53 classification Roads)
.Other: LongInt (includes all A61, A62, A64, A73 and A74 classification Roads)

For example, ***.OptiRouter.Speed.Interstate = 65*** would set the speed limit for the Interstate highways, while performing the OptiRouter calculations, to 65 mi/hr. ***.OptiRouter.Speed.Priority = 1*** would give interstate highways a selection priority of 1, when thr OptiROuter is attempting to decide which road to select for the route calculations.

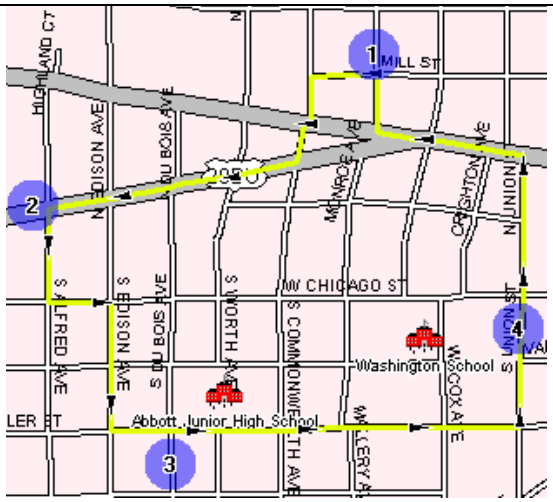
VB Example
```
Private Sub Command14_Click()
  ' Set Selection priorities for the various road types.
  MapPro1.OptiRouter.Priority.Interstate = 1
  MapPro1.OptiRouter.Priority.Primary = 1
  ' Set Secondary Roads the preferred ones
  MapPro1.OptiRouter.Priority.Secondary = 6
  MapPro1.OptiRouter.Priority.Road = 1
  MapPro1.OptiRouter.Priority.Trail = 1
  MapPro1.OptiRouter.Priority.Other = 1
End Sub
```

.OptiRouter.RoutePath Interface

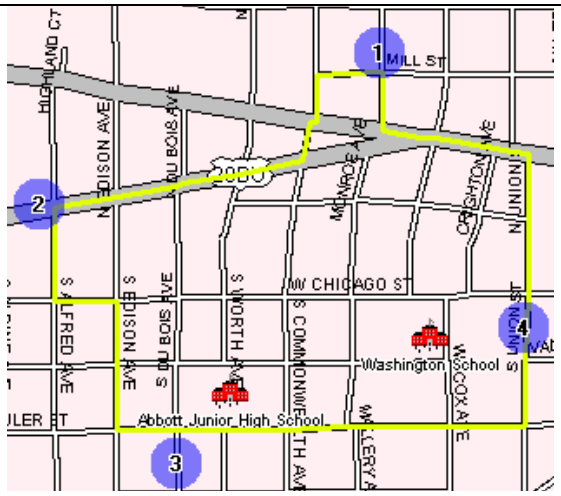
This interface provides the user with the capabilities of setting the way the calculated optimized route will be displayed, as well as other visual parameters.

.ArrowVis: Boolean
Display Travel direction arrows on the calculated Route.

.RouteVis: Boolean
Display/Highlight the calculated Route.



With Direction Arrows



Without Direction Arrows

.RouteColor: LongInt
Set the color of the Route highlight

.RouteOpacity: LongInt
Set the opacity level (from 0=Solid to 100= Clear) of the Route highlight

.RouteWidth: LongInt
Set the Width of the route highlight, in pixels.

.RouteLineStyle: LongInt
Specify the line type to be used for routing. This is here for future expansion. Path routes are highlighted with transparent solid lines of the width and color specified by the user. The transparency of the highlighting is set using the .RouteOpacity property.

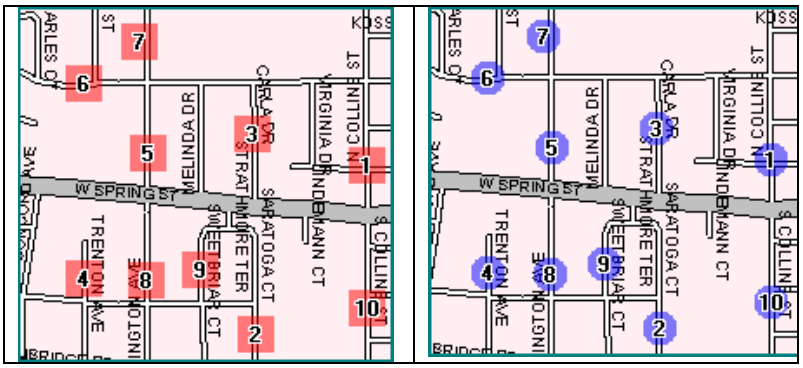
.MarkVis: Boolean
Display the Markers used to identify the points you have set for the route optimization.

.MarkColor: LongInt
Set the color of the Markers used to identify the points you have set for the route optimization.

.MarkSize: LongInt
Set the size of the Markers used to identify the points you have set for the route optimization (this is actually the size of the circumscribed square), in pixels.

.MarkType: LongInt
Specify the Mark type to be used in identifying the points you have set for the route optimization.

The available types of markers are 0-Square, and 1-Round. The squares are painted in a semi-transparent mode, and the point number is displayed in the middle of the marker, if the appropriate properties have been set.



.MarkColor: LongInt
Set the color of the Markers used to identify the points you have set for the route optimization.

.FontVis:Boolean
Sets the Visibility of the Text used to identify the routing points set by the user. This is controlled independently of the marker available for each point.

.FontName:String
Sets the font to be used for the Text used to identify the routing points set by the user.

.FontSize:Integer
Sets the Point size of the Text used to identify the routing points set by the user.

.FontBold:Boolean
Sets the Bold state of the Text used to identify the routing points set by the user.

VB Example

```
Private Sub Command1_Click()
  MapPro1.OptiRouter.RoutePath.RouteVis = True
  MapPro1.OptiRouter.RoutePath.RouteColor = vbRed
  MapPro1.OptiRouter.RoutePath.RouteWidth = 10
  MapPro1.OptiRouter.RoutePath.FontSize = 14
  MapPro1.OptiRouter.RoutePath.FontBold = True
  MapPro1.OptiRouter.RoutePath.MarkVis = True
  MapPro1.OptiRouter.RoutePath.MarkColor = vbBlue
  MapPro1.OptiRouter.RoutePath.MarkSize = 10
  MapPro1.OptiRouter.RoutePath.MarkType = 3
End Sub
```

.OptiRouter.RouteParse(index:Integer; s:string):String — Function

Returns the desired portion of S, where S is the composite string returned by .OptiRouter.GetNextRoute. Index can be either the numeric order of the desired field in the composite string, or the actiual name of the field.

Index	Field
1	Leg
2	Type
3	Instructions
4	Street
5	Place
6	STime
7	SDistance
8	Time
9	Distance
10	Speed
11	Lon or X
12	Lat or Y

VB Example

```
Private Sub GetFirstNext_Click()
  ' Calculate the optimized route using already defined points
  MapPro1.OptiRouter.Calculate (0)
  ' Get the segments and populate the Listbox
  Rseg = MapPro1.OptiRouter.GetRouteFirst
  ' Chek to see if the end of the list object has been reached
  While Rseg <> ""
  Rseg_1 = MapPro1.OptiRouter.RouteParse("Leg", Rseg)
  Rseg_2 = MapPro1.OptiRouter.RouteParse("Street", Rseg)
  Rseg_3 = MapPro1.OptiRouter.RouteParse("Distance", Rseg)
  Rseg_4 = MapPro1.OptiRouter.RouteParse(10, Rseg)
    List1.AddItem "Leg: " & Rseg_1 & ", Street: " & Rseg_2 & ",
Dist: " & Rseg_3 & ", Time: " & Rseg_4
    Rseg = MapPro1.OptiRouter.GetRouteNExt
  Wend
  ' Close the List object to release
  MapPro1.OptiRouter.GetRouteClose
End Sub
```

.OptiRouter.RouteType:TxRouteType — Property

Enumerated list of the route type calculation the user is interested in.

0-Shortets
1-Fastest
2-Direct
3-Preferred
4-ShortUnbiased

VB Example

```
Private Sub Command17_Click()
' Set the route type to use in the calculations
  MapPro1.OptiRouter.RouteType = Direct
  MapPro1.OptiRouter.Calculate 1
End Sub
```

.OptiRouter.Speed:RoadRec — Property

Uses a RoadRec record structure to set the speed limits (mi/hr) for the six main road types.

VB Example

```
Private Sub Command16_Click()
  ' Set Speed Limits for the various road types.
  MapPro1.OptiRouter.Speed.Interstate = 65
  MapPro1.OptiRouter.Speed.Primary = 55
  MapPro1.OptiRouter.Speed.Secondary = 45
  MapPro1.OptiRouter.Speed.Road = 30
  MapPro1.OptiRouter.Speed.Trail = 5
  MapPro1.OptiRouter.Speed.Other = 25
End Sub
```

Delphi Example

```
procedure TForm1.SMPSpEdit3Change(Sender: TObject);
// Use custom SpinEdit control to set Speeds
var t:integer;
begin
  t:=tsmpspedit(sender).tag;
  With MapPro1.OptiRouter.Speed do
  begin
    Case t of
     0:Interstate:=smpspedit1.ivalue;
     1:Primary:=smpspedit2.ivalue;
     2:Secondary:=smpspedit3.ivalue;
     3:Road:=smpspedit4.ivalue;
     4:Trail:=smpspedit5.ivalue;
     5:Other:=smpspedit6.ivalue;
    end;
  end;
end;
```

.OptiRouter.SaveToFile… — Future

Reserved for possible future expansion of the module's capabilities.

.OptiRouter.SavePoints(s:String) — Method

Saves the currently specified routing points set by the user, to the specified text file. The text file format is one line per specified point, with each line containing:

Name, X, Y

VB Example

```
Private Sub Command18_Click()
' Save currently defined Points for later use
  MapPro1.OptiRouter.SavePoints "MyTestOptiPts2.pts"
End Sub
```

.ImportLayer Interface

An simple interface that enables the user to import a single MID/MIF and SHP/DBF file pair and overlay it on the MapPro map surface. Note that ***no*** data translation takes place, i.e., that data is NOT converted to the native MapPro format, it is simply painted on the map as an overlay. Also note that the data has to be in X,Y projection – no other projections are supported. This interface was implemented in earlier releases and it was left active for compatibility with older products using the MapPro71.OCX. In Release 2, a much more powerful interface was added, that allows the importing of multiple MID/MIF and/or shape files. See the section on the **ImportMgr** interface for more details.

Any attributes in the imported files are ignored and a ***single set*** of Pen, Brush, Mark and Font attributes are assigned to all elements in the imported file, by the user.

Importing such data files can be achieved either

(d) Programatically, by directly calling a sequence of Import.Interface methods, or

(e) By calling the method Import.ShowDialog and then using the options within that dialog.

The data from the specified file is not actually loaded into memory, but rather is played onto the map when the map is redrawn, thus allowing much larger external data sets to be rendered.

The methods and properties that are available through the Import interface are explained in this section. Note that instead of using **ImportLayer**, as the root in the examples given below, the property off the main control, **Import**, is used since that's how it would be called from the application.

.Import.Brush **Property**

Holds the brush definition that is used to paint (fill) the interiors of polygon objects, circles, etc.) The Default brush is Solid, White, Opaque.

.brush.BackColor (N/A)
This property was kept in the brush interface to keep it consistent with other brush structures used in the control (e.g., the CAD.brush structure), but has no effect at this time.

.brush.Color
Defines the brush color

.brush.Mode (N/A)
This property was kept in the brush interface to keep it consistent with other brush structures used in the control (e.g., the CAD.brush structure), but has no effect at this time.

.brush.Style
Constants specifying the brush style (note that the actual line style is controlled by the pen attribute).

0 – Solid
1 – Clear

2 – Horizontal
3 – Vertical
4 – Left Dash
5 – Right Dash
6 – Cross
7 – Diagonal Cross

.brush.Visible
Sets the Visibility of the brush.

VB Example
```
Private Sub Command126_Click()
 ' Set the Upper and Lower visibility thresholds
 MapPro1.Import.Upper = 5000
 MapPro1.Import.Lower = 5
 ' Set brush color (this would be the fill color)
 MapPro1.Import.Brush.Color = vbYellow
 ' Set pen color (for polygon outline)
 MapPro1.Import.Pen.Color = vbBlue
 ' Set hatch style
 MapPro1.Import.Brush.Style = 4
 ' setting MapPro1.Import.Brush.Visible = False would
 ' only print the polygon outline
 ' Import a shape file (and the associated DBF file)
  MapPro1.Import.FileName = "D:\country_col_region.shp"
 ' Redraw the map to reflect the imported data
 MapPro1.Redraw
End Sub
```

.Import.Count:Integer — Property

The number of objects in the imported file.

Delphi Example
```
procedure TForm1.Button16Click(Sender: TObject);
var i,j,n:integer;
begin
  // Cycle through the imported layers and show # of objects
  If Mappro1.ImportMgr.Count > 0 then
  begin
    for i:=1 to Mappro1.ImportMgr.Count do
    begin
      listbox2.items.add('No. '+inttostr(i)+
       'L: ' + Mappro1.ImportMgr.items[i-1].name+
       'F. ' + Mappro1.ImportMgr.items[i-1].FileName+
       ' #. '+ inttostr(Mappro1.ImportMgr.items[i-1].count));
    end;
  end else label1.caption:='No Layers have been imported';
end;
```

.Import.Filename:String — Property

Specifies the Filename that is to be imported. If a filename is specified and the Visibility is set to True, then the data is automatically rendered (subject to visibility scale thresholds, etc.), no further action is required. The filename needs to also include an extension of either ".MIF" or ".SHP". Note that all imported files need to be in X-Y projection. No other projections are supported at this time.

VB Example

```
Private Sub Command11_Click()
' Import a shape file (and the associated DBF file)
' No additional info is needed to import the data (using
' the built-in default attributes.
  MapPro1.Import.FileName = "D:\TG24001.shp"
 ' Redraw the map to reflect the imported data
 MapPro1.Redraw
End Sub
```

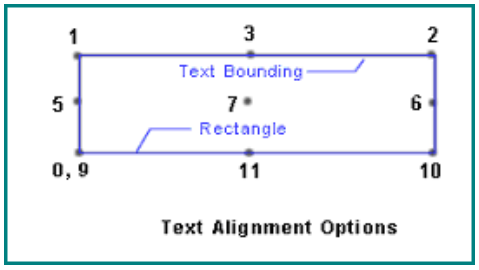
.Import.Font — Property

This interface specifies the Font attributes to be used when labeling imported objects. Note that the property *Import.LabelFilter* is used to specify what type of objects are to be labeled.

.font.Align: Integer

Controls the alignment of the text within the text bounding rectangle. Note that the reference point of any text string is the bottom left corner of the string.

The default vertical aligment is zero, which sets the top of the text bounding rectangle aligned with the reference point. There are basically 3 primary top alignment options, 1=Left, 2=Right, 3-Center. Those options *"**OR**ed with 4*" also provide *"middle"* vertical alignment, while *"**OR**ed with 8"* provide *"bottom"* vertical alignment.



.font.Angle:Integer (N/A)

This property was kept in the font interface to keep it consistent with other font structures used in the control (e.g., the CAD.font structure), but has no effect at this time.

.font.BackColor:Integer (N/A)

This property was kept in the font interface to keep it consistent with other font structures used in the control (e.g., the CAD.font structure), but has no effect at this time.

.font.Color:Integer

Sets the foreground Font Color.

.font.Height:Integer
Sets the Font height (pixels). You may alss set the Font Size in points (See below).

.font.Mode:Integer (N/A)
This property was kept in the font interface to keep it consistent with other font structures used in the control (e.g., the CAD.font structure), but has no effect at this time.

.font.Name:String
Specify the name of the font (as it appears in the OS font listing)

.font.Size:Integer
Sets the font size in points. You may alss set the Font Height in pixels (See above).

.font.Style:Integer
Three basic font styles are available, as shown beloa

Style #	Font Style
1	Bold
2	Italic
3	Bold Italic

.font.Visible:Boolean
Sets the visibility of the ltext used to label objects.

VB Example

```
Private Sub Command127_Click()
 ' Set the Upper and Lower visibility thresholds
 MapPro1.Import.Upper = 2000
 MapPro1.Import.Lower = 1
 ' Set the color of the pen and the font for the labels
 MapPro1.Import.Pen.Color = vbGreen
 MapPro1.Import.Font.Color = vbBlue
 ' Set the Font Size and Style (3 = Bold Italic)
 MapPro1.Import.Font.Size = 12
 MapPro1.Import.Font.Style = 3
 ' Set the font name
 MapPro1.Import.Font.Name = "Arial Black"
 ' Set the label Filter to Points, Lines & Polys
 MapPro1.Import.LabelFilter = 1 + 2 + 4
 MapPro1.Import.Font.Visible = True
 ' Set the First field to be used for lab
 MapPro1.Import.LabelField = 1
 ' Import a shape file (and the associated DBF file)
MapPro1.Import.FileName = "D:\country_col_region.shp"
 ' Redraw the map to reflect the imported data
 MapPro1.Redraw
End Sub
```

.Import.LabelField:Variant **Property**

Specifies which field in the database will be used to label the imported objects. The field #, or the field name (if known) can be used.

VB Example

```
Private Sub Command17_Click()
' Set the Upper and Lower visibility thresholds
 MapPro1.Import.Upper = 2000
 MapPro1.Import.Lower = 1
‘ Use the first field to label the objects
 MapPro1.Import.LabelField = 1
' Import a MID/MIF pair
 MapPro1.Import.FileName = "D:\TGR24003.MIF"
' Redraw the map to reflect the imported data
 MapPro1.Redraw
End Sub
```

.Import.LabelFilter:Integer — Property

Specifies which types of objects to be labeled when imported, using a bit position approach (so that options can be combined).

0 – Do not Label anything
1 – Label Points (for text alignment see the .Font interface)
2 – Label Lines (label is parallel to segment at midpoint)
4 – Label Polygons (Polygon centroid is labeled)

For example, setting the value of this property to 3 would label points and lines.

Note that when points and polygon centroids are labeled, the labels are horizontal, and there is some collision detection that is performed to eliminate possible interference of the labels. Polylines, however, which are labeled paralled to the segment that the label corresponds to, have no collision detection, so care should be taken to set approximate Upper and Lower values to eliminate some of the possible labeling clutter.

.Import.Lower — Property

The lower threshold of visibility for the imported data (miles/km – depending on unts that have been selected. For example, if Lower=2, then when the scale factor is below 2 miles/km, the imported data is not rendered.

VB Example

```
Private Sub Command42_Click()
' Private Sub Command125_Click()
 ' Set the Upper and Lower visibility thresholds
 MapPro1.Import.Upper = 50
 MapPro1.Import.Lower = 5
 ' Import a shape file (and the associated DBF file)
 ' This is the only action needed to import the file
‘ using the default attributes
 MapPro1.Import.FileName = "N:\TGR25001A.SHP"
 ' Redraw the map to reflect the imported data
 MapPro1.Redraw
End Sub
```

Interface containing the attributes for the mark that is used to visualize point data, or mark polygon centroids.

.mark.BackColor
The color of the pen used to draw the perimeter of the mark.

.mark.Handle
The handle of a bitmap, obtained by the user, that will be used to mark points and polygon centroids, if Mark.Style is set to zero.

.mark.Color
The brush color used to flood the interior of the mark.

.mark.Size
The size of the Mark in pixels.

.mark.Style
Specifies the type of mark to be used. 0 = Reserved, 1 = Square marker, 2 = Elipse (circular) marker. If a negative number is specified, then it uses the bult-in CAD marker corresponding to the absolute value of the variable (CAD Markers are reference row-wise). If Reserved is specified, then the Mark.Handle is used to get the mark to be used for the points.

.mark.Transparent:Boolean
This property is used to set the transparency of the marks used for points and polygon centroids. It only has an effect if Mark.Style is set to 0 or a negative value. If Transparent=True, then the color of the lower left pixel of the user specified bitmap is used as the transparent color.

.mark.Visible
Sets the visibility of markers in the imported point data.

VB Example

```
Private Sub Command127_Click()
'-------
  ' Here a handle is used to get one of the build-in marks,
  ' for demo purposes.  If just a CAD.mark was desirable,
  ' the user could have simply set the style to -2, as well.
  hmark = MapPro1.Cad.GetMarker(2)
  MapPro1.Import.Mark.Handle = hmark
  MapPro1.Import.Mark.Style = 0
'-------
  ' Set the MArk transparency
  MapPro1.Import.Mark.Transparent = True
  'Set visibility thresholds
  MapPro1.Import.Upper = 3000
  MapPro1.Import.Lower = 0.5
  ' Set the color of the pen and the font for the labels
  MapPro1.Import.Pen.Color = vbGreen
  MapPro1.Import.Font.Color = vbRed
  ' Set the Font Size and Style
  MapPro1.Import.Font.Size = 11
  MapPro1.Import.Font.Style = 1
  ' Set the font name
```

```
  MapPro1.Import.Font.Name = "Arial"
  ' Set the label Filter to Polys
  MapPro1.Import.LabelFilter = 4
  MapPro1.Import.Font.Visible = True
  ' Set the First field to be used for lab
  MapPro1.Import.LabelField = 1
  ' Import a shape file (and the associated DBF file)
  ' MapPro1.Import.FileName = "N:\trius\smpmap\TGR25001A.SHP"
  MapPro1.Import.FileName = "D:\Develop\undertow\maptivate\sample-
source-VB\country_col_region.shp"
  ' Redraw the map to reflect the imported data
  MapPro1.Redraw
End Sub
```

.Import.Name:String **Property**

A name assigned to the imported layer by the user. This name appears in the ImportMgr Dialog and may also be specified through the Import.Dialog interface. Note that when a file is imported, the value of this property is automatically set to the filename, as well, until modified by the user.

.Import.Opacity:Integer **Method**

Controls the Opacity of the brush used to paint the interior of polygons. It can take values from 0 to 100. A value of zero corresponds to a totally transparent (clear) polygon, and a value of 100 corresponds to a totally opaque (solid) polygon.

It should be pointed out that this Opacity property only affects the map rendering on th e screen. The Opacity property is NOT supported for printed output. All polygon fills in printed output are generated as if the opacity value was 100.

.Import.Pen **Method**

Sets the attributes for the pen to be used to paint imported line objects.

.pen.BackColor (N/A)
This property was kept in the pen interface to keep it consistent with other pen structures used in the control (e.g., the CAD.pen structure), but has no effect at this time.

.pen.Color
The pen color to be used for lines and polygon outlines.

.pen.Mode (N/A)
This property was kept in the pen interface to keep it consistent with other pen structures used in the control (e.g., the CAD.pen structure), but has no effect at this time.

.pen.ROP
This property was kept in the pen interface to keep it consistent with other pen structures used in the control (e.g., the CAD.pen structure), but has no effect at this time. There is no Need for the ROP, since the Opaque property of the Import interface achieves the same result.

.pen.Style
Set the style of the pen used to draw the lines of fill patterns. Note that this style is meaningful ONLY when the pen width is set to 0.

0	Solid
1	Dash
2	Dot
3	Dash-dot
4	Dash-dot-dot
5	Invisible
6	Inside solid

.pen.visible
Sets the visibility of the pen (therefore, of the line objects, painted by the pen).

.pen.width
Pen width in pixels. (Note that pen widths other than 1 do not support any of the pen styles – only solid).

VB Example

```
Private Sub Command127_Click()
  MapPro1.Import.Mark.Style = 2
  MapPro1.Import.Mark.Size = 10
  MapPro1.Import.Mark.BackColor = vbBlue
  MapPro1.Import.Mark.Color = vbYellow
  'Set visibility thresholds
  MapPro1.Import.Upper = 3000
  MapPro1.Import.Lower = 0.5
  ' Set the color of the pen and the font for the labels
  MapPro1.Import.Pen.Color = vbBlue
  MapPro1.Import.Pen.Width = 4
  MapPro1.Import.Font.Color = vbRed
  ' Set the Font Size and Style
  MapPro1.Import.Font.Size = 11
  MapPro1.Import.Font.Style = 1
  ' Set the font name
  MapPro1.Import.Font.Name = "Arial"
  ' Set the label Filter to Polys
  MapPro1.Import.LabelFilter = 7
  MapPro1.Import.Font.Visible = True
  ' Import a MID/MIF pair
  MapPro1.Import.FileName = "D:\TEMP\L\L9995.MIF"
```

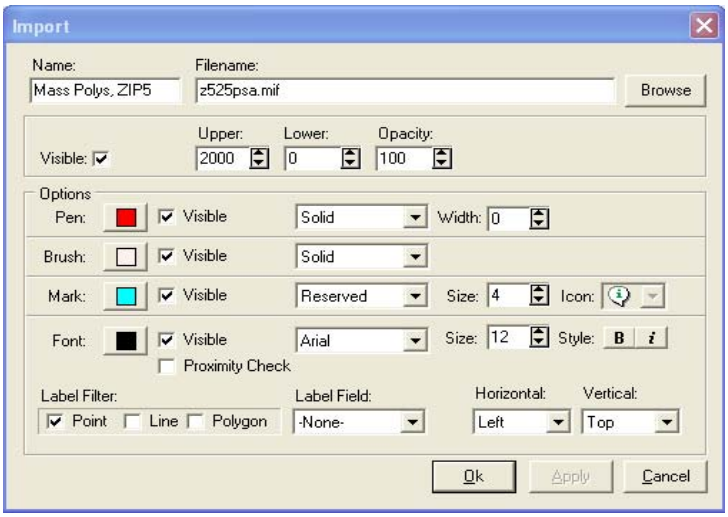
```
    ' Redraw the map to reflect the imported data
    MapPro1.Redraw
End Sub
```

.Import.ProximityCheck:Boolean — Method

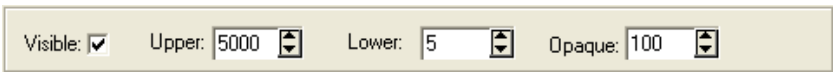
Determines whether imported objects will be labeled automatically and indiscriminately, or a proximity check will be performed and no overlapping labeling will be permitted.

.Import.ShowDialog — Method

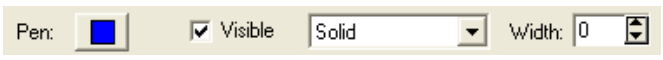
Opens up a dialog that allows the user to dynamically specify the data file to be imported, as well as the attributes to be used when rendering the dta onto the map.



Each section of the dialog is briefly described below:



Permits the user to set the visibility, Upper and Lower visibility thresholds and the opacity of the brush used to fill polygons.



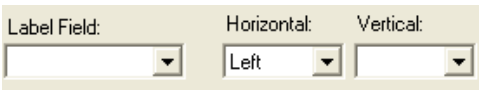
Permits the user to set the pen attributes (for drawing line objects and the outline of polygon objects)

Permits the user to set the attributes of the brush used to fill polygon objects.

Permits the user to select the type of mark to be used for labeling point opjects and polygon centroids. Note that if "icon" is selected, then the user may select from a list of built-in bitmap icons.

Permits the user to set the attributes of the font user to label imported objects.

Permits the user to select whether to label Points, Lines, and Polygon (centroids), using the font attributes set above.



Permits the user to select the field (from the imported database) to be used for labeling, and to set the horizontal and vertical text alignment of the labels.

VB Example

```
Private Sub Command121_Click()
  ' A single call to the method is needed.  Note that since this is
an interface,
  ' in VB you have to set the result to a variable.
  s = MapPro1.Import.ShowDialog
End Sub
```

.Import.Upper — Property

The upper threshold of visibility for the imported data (miles/km – depending on unts that have been selected. For example, if Upper=100, then when the scale factor is above 100 miles/km, the imported data is not visible.

VB Example

```
Private Sub Command42_Click()
```

```
' Private Sub Command125_Click()
 ' Set the Upper and Lower visibility thresholds
 MapPro1.Import.Upper = 50
 MapPro1.Import.Lower = 5
 ' Import a shape file (and the associated DBF file)
 ' This is the only action needed to import the file using the
default
 ' attributes
 MapPro1.Import.FileName = "N:\TGR25001A.SHP"
 ' Redraw the map to reflect the imported data
 MapPro1.Redraw
End Sub
```

.Import.Visible:Boolean **Property**

Sets the visibility of the imported data (note that the visibility of points, lines and polygons, individually, may also be controlled using the visibility property of the mark, pen and brush. Also note that this is setting takes precedence over the Upper and Lower settings, if it set to false.

.ImportManager Interface

An new, powerful import interface that enables the user to import multiple MID/MIF and SHP/DBF file pairs and overlay them on the MapPro map surface. Note that ***no*** data translation takes place, i.e., that data is NOT converted to the native MapPro format, it is simply painted on the map as an overlay. Also note that the data has to be in X,Y projection – no other projections are supported, at this point.

Any attributes in the imported files are ignored and attributes for each such layer may be assigned by the user.

Importing such data files as layers can be achieved either

(f) Programatically, by directly calling a sequence of ImportMgr Interface method, or

(g) By calling the method ImportMgr.ShowDialog and then using the options within that dialog.

The data from the specified files is not actually loaded into memory, but rather is played onto the map when the map is redrawn, thus allowing much larger external data sets to be rendered.

The methods and properties that are available through the ImportMgr interface are explained in this section. Note that instead of using **ImportManager**, as the root in the examples given below, the property off the main control, **ImportMgr**, is used since that's how it would be called from the application.

.ImportMgr.AddLayer(Name:WideString, FileName:Widestring) Method

---

Adds the layer specified by **FileName** to the Import Manager, and assigns **Name** to it. The Layer is appended at the bottom of the currently loaded layers and the layers count is incremented by one.

Delphi Example

```
procedure TForm1.Button6Click(Sender: TObject);
var n:integer;
begin
  // Import the file Z25P.MIF and name it "Mass Points, ZIP5"
  Mappro1.ImportMgr.AddLayer('Mass Points, ZIP5','z25p.shp');
   // The file just loaded is a points file, set some attributes
  with MapPro1.ImportMgr do
  begin
    currentindex:=0;
    currentlayer.Font.Color:=clgreen;
    currentlayer.Font.size:=24;
    currentlayer.LabelField:=2;
    currentlayer.labelfilter:=1;
    currentlayer.Mark.Style:=4;
    // Now use current layer to set the opacity to 30%
    currentlayer.Opacity:=30;
  end;
  // We'l also load a polygons shape file Z525PSA.mif
```

```
  Mappro1.ImportMgr.AddLayer('Mass Polys, ZIP5','z525psa.mif');
  // Zoom the viewport to the extents of the imported layers
  mappro1.ImportMgr.ZoomLayers(-1);
end;
```

.ImportMgr.Clear — Method

Clears all imported layers.

Delphi Example

```
procedure TForm1.Button7Click(Sender: TObject);
begin
  Label1.caption:=inttostr(mappro1.importmgr.count);
  Mappro1.ImportMgr.clear;
  // Check to see that the count is zero
  Label2.caption:=inttostr(mappro1.importmgr.count);
end;
```

.ImportMgr.Count:Integer — Property

The total number of layers currently imported.

Delphi Example

```
procedure TForm1.Button7Click(Sender: TObject);
begin
  // Show # of layers before and after clear is called
  Label1.caption:=inttostr(mappro1.importmgr.count);
  Mappro1.ImportMgr.clear;
  // Check to see that the count is zero
  Label2.caption:=inttostr(mappro1.importmgr.count);
end;
```

.ImportMgr.CurrentIndex:Integer — Property

The currently selected Layer. Makes it easy for the user to access the Layer info using the ImportMgr.CurrentLayer interface, once the CurrentIndex is set.

Delphi Example

```
procedure TForm1.Button6Click(Sender: TObject);
begin
  // Import the file Z525PSA.MIF and name it "Mass, ZIP5"
  Mappro1.ImportMgr.AddLayer('Mass, Zip5','z525psa.mif');
  // The file just loaded is a polygon file, set some attributes
```

```
  with MapPro1.ImportMgr do
  begin
    // set the current layer index
    currentindex:=0;
    // Now use current layer to set the opacity to 30%
    currentlayer.Opacity:=30;
    // Set the brush color and style (cross)
    currentlayer.Brush.Color:=clblue;
    currentlayer.Brush.Style:=6;
  end;
  mappro1.Redraw;
end;
```

.ImportMgr.CurrentLayer:ImportLayer Property

The current default layer inerface. Makes it easier for the developer to access the properties of the current layer without having to explicitly use the ImportMgr.Items array.

Delphi Example

```
procedure TForm1.Button6Click(Sender: TObject);
begin
  // Import the file Z525PSA.MIF and name it "Mass, ZIP5"
  Mappro1.ImportMgr.AddLayer('Mass, Zip5','z525psa.mif');
  // The file just loaded is a polygon file, set some attributes
  with MapPro1.ImportMgr do
  begin
    // set the current layer index
    currentindex:=0;
    // Now use current layer to set the opacity to 30%
    currentlayer.Opacity:=30;
    // Set the brush color and style (cross)
    currentlayer.Brush.Color:=clblue;
    currentlayer.Brush.Style:=6;
  end;
  mappro1.Redraw;
end;
```

.ImportMgr.DeleteIndex(n:integer) Method

Deletes the N-th layer from the list (the layers indexed array is zero-based), and decreases the property ImportMgr.Count.

Delphi Example

```
procedure TForm1.Button8Click(Sender: TObject);
begin
  // Delete the 2nd layer, remembr, array is 0-based
  mappro1.ImportMgr.DeleteIndex(1);
  // Open dialog to confirm
```

```
  mappro1.ImportMgr.ShowDialog;
end;
```

.ImportMgr.DeleteNamedLayer(Lname:string):Integer — Method

Deletes the specified named layer and returns the index # of the layer that was deleted, or -1 if the deletion process failed (because no such layer was found).

Delphi Example

```
procedure TForm1.Button8Click(Sender: TObject);
var n:integer;
    s:string;
begin
  // Delete a named Lyer
  s:='Second Layer';
  n:=mappro1.ImportMgr.DeleteNamedLayer(s);
  if n=-1 then label1.caption:=s+' - was not found'
    else label1.caption:=s+' - Deleted';
  // Open dialog to confirm
  mappro1.ImportMgr.ShowDialog;
end;
```

.ImportMgr.Items[n]:ImportLayer — Property

Array of loaded Layer interfaces. The user can access any of the Layers through this indexed array.

Delphi Example

```
procedure TForm1.Button7Click(Sender: TObject);
begin
  // Get the Layers Count.  If count>1 then access and modify
  // the second laye attributes
  If Mappro1.ImportMgr.Count>1 then
  begin
    Mappro1.ImportMgr.Items[1].Pen.color:=clyellow;
    Mappro1.ImportMgr.Items[1].Pen.width:=6;
    // Echo the number of object in this imported file
    Label2.caption:=inttostr(Mappro1.ImportMgr.Items[1].count);
  end;
end;
```

.ImportMgr.LoadfromFile(Fname:String) — Method

Loads a previously saved import layer configuration file (*.ILM).

Delphi Example

```
procedure TForm1.Button9Click(Sender: TObject);
```

```
begin
  // load Layers import config file
  mappro1.ImportMgr.LoadFromFile('Mysample.ilm');
end;
```

.ImportMgr.SaveToFile(Fname:String) Method

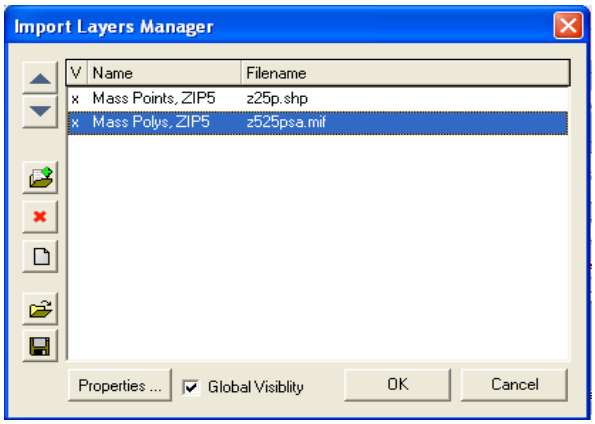
Saves the current Import layer configuration to an external file, with the extension .ILM that can be loaded later on, if desired.

Delphi Example

```
procedure TForm1.Button9Click(Sender: TObject);
begin
  // Save Import Layers config file - extension needed
  mappro1.ImportMgr.savetoFile('Mysample2.ilm');
end;
```

.ImportMgr.ShowDialog Method

It opens the Import Manager dialog that allows the user to manage all the import layers, their properties, etc.



The various parts of the Import Manager dialog, the informations displayed and the actions available are described in detail below.

The three columns of information displayed is the Visibility state of a layer, the name assigned to the layer by the user, and the filename containing the data for the imported layer. Note that clicking on the "X" in the visibility column, toggles the visibility of the selected layer On/Off. It should also be noted that the visibility of each layer is also controlled by the individual visibility thresholds for the selected layer, which can be set through the "Properties" button.

Move the selected layer on position up in the list. This is very important because layers are painted in the order they appear in this list, i.e., the layer that appears at the top of the list is painted first, then the second layer is painted on top of that, and so on.

Move the selected layer on position Down in the list.

Add a layer to the list. It opens up a stock windows file dialog and allows the user to select a MIF

or a SHP file to add as a layer.

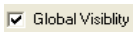
Delete the currently highlighted layer.

Clear (Delete) All layers.

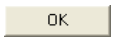
Open/Load a previously saved Import Layers configuration (*.ILM) file.

Save the current Layer(s) Definition. It opens up a stock windows dialog and allows you to save the current Import Layers configuration to a file, with the extension "ILM", that can be opened/loaded at a later time.

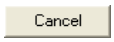
Opens the Properties dialog (see .Import.Dialog) and allows the user to set the properties for the currently selected Layer.



Sets the visibility of ALL layers in the list (Takes precedence over the Upper/Lower visibility settings of individual layers)



Saves all changes and closes the dialog



Cancels any changes made during this session, and closes the dialog.

Delphi Example

```
procedure TForm1.Button7Click(Sender: TObject);
begin
  // Display the Import Layers Manager Dialog
  Mappro1.importMgr.ShowDialog;
end;
```

.ImportMgr.Visible:Boolean Property

Sets the visibility of the Layer Manager interface, i.e., all imported layers.

.ImportMgr.ZoomLayers(N:Integer) Method

Zooms out to an imported layer extents, i.e., so that the extents of the specified imported Layer, N, fill up the viewport. If it is called with an agument of N = -1, it zooms in/out so that ALL imported layers are visible in thme viewport.

Delphi Example

```
procedure TForm1.Button7Click(Sender: TObject);
begin
  // Display the Import Layers Manager Dialog
  Mappro1.importMgr.ShowDialog;
End;
```

.POIManage Interface

This interface is usable only if the developer has licensed and is using the premium Points Of Interest (POI) dataset. It allows the developer to programmatically control the searching and rendering of the database points. See Appendix N and Appendix O for details on licensing, deployment and pricing of the POI data.

The categories and subcategories of the POIs in this datavase are provided below. (You can see how they are actually used later in this section).

Category Code	Category Name	Sub-Category Name
500100	Automobile Club	N (N – denotes none)
500101	Towing Service	N
500202	Car Parking	Parking Garage
500300	Car parts & accessories	N
500400	Car Rental	N
500500	Car Repair facility	N
500600	Car Repair/Dealer	N
500700	Car Wash	N
500800	Petrol/Gas Station	N
500900	Motorcycle Repair/Dealer	N
501000	Boat Repair/Dealer	N
502000	Recreational Vehicles/Dealer	N
510100	Bank	N
510300	Convention Center	N
510400	Currency Exchange	N
530200	School	N
530201	School	Nursery school
540100	Pharmacy	N
540201	Doctor	General Practitioner
540202	Doctor	Specialist
540210	Dental Surgeon/Dentist	N
540220	Veterinarian	N
540400	Emergency Medical Service	N
540500	Fire Station	N
540700	Hospital/Polyclinic	N
540800	Police Station	N
540801	Police Station	Municipal
540803	Police Station	State
550100	Art Gallery	N
550200	Arcade	N
550300	Casino	N
550400	Cinema	N
550500	Museum	N
550600	Night Life	N
550601	Night Life	Discotheque
550700	Stage	N
550703	Stage	Cultural center
550706	Stage	Theater
550800	Winery & Brewery	N

560100	Fast food	N
560200	Bar	N
560201	Bar	Microbrewery/Beer Garden
560202	Bar	Cocktail Bar
560300	Ice cream parlor	N
560400	Pizzeria	N
560500	Restaurant	N
560501	Restaurant	Asian (other)
560502	Restaurant	American
560503	Restaurant	Barbecue
560504	Restaurant	CafÃ© & Espresso
560505	Restaurant	Chinese
560506	Restaurant	Continental
560507	Restaurant	Creole-Cajun
560509	Restaurant	French
560510	Restaurant	Greek
560511	Restaurant	Indian
560512	Restaurant	Italian
560513	Restaurant	Japanese
560515	Restaurant	Mexican
560516	Restaurant	Seafood
560517	Restaurant	Steak Houses
560518	Restaurant	Thai
560519	Restaurant	German
560520	Restaurant	Spanish
560521	Restaurant	Vietnamese
560523	Restaurant	Korean
560524	Restaurant	Jamaican
560525	Restaurant	Hawaiian
560526	Restaurant	Polish
570200	Court House	N
570400	Government Office	N
570401	Government Office	Municipal
570402	Government Office	County or equivalent
570403	Government Office	State
570404	Government Office	National
570405	Government Office	Supra National
580100	Camping	N
580200	Bed&Breakfast	N
580300	Hotel	N
580500	Recreational Camp	N
580600	Youth Hostel	N
590100	Travel agency	N
600100	Amusement Park	N
600300	Fairground	N
600500	Park	N
600550	Zoo	N
600551	Zoo	Aquarium
600600	Stadium	N
600604	Stadium	Horse racing
600610	Stadium	Motor Sport
600700	Thematic Sport	N

600701	Thematic Sport	Bowling alley
600702	Thematic Sport	Golf Course
600703	Thematic Sport	Skating rink
600705	Thematic Sport	Swimming pool
600706	Thematic Sport	Tennis court
600707	Thematic Sport	Yacht Basin/Marina
600708	Thematic Sport	Squash court
600709	Thematic Sport	Billiard Parlor/Pool Hall
600800	Thematic Outdoor Sport	N
600804	Thematic Outdoor Sport	Ski resort
600900	Fitness Club	N
610100	Library	N
610200	Post Office	N
610400	Tourist Information Office	N
620100	Cash dispenser/ATM	N
620200	Shopping Center	N
620300	Shopping Service	N
620301	Shopping Service	Beauty salon
620302	Shopping Service	Barber shop
620304	Shopping Service	CD/Video rental
620305	Shopping Service	Laundry
620306	Shopping Service	Photo lab/Development
620307	Shopping Service	Photocopy
620400	Shop	N
620401	Shop	Antique/Art
620402	Shop	Beauty
620403	Shop	Books/Magazines
620404	Shop	Camera/Clocks/Wristwatch
620405	Shop	Computer/Consumer electronics
620406	Shop	CD/Video shop
620407	Shop	Hardware/Home improvement
620408	Shop	Drug store
620409	Shop	Electric appliance
620410	Shop	Fashion
620411	Shop	Flower shop
620412	Shop	Hobby/Free Time
620413	Shop	Furniture/Home Furnishing
620414	Shop	Gardening
620415	Shop	Glassware/Ceramic
620416	Shop	House/Office
620417	Shop	Jeweler
620419	Shop	Opticians shop
620421	Shop	Recycling shop
620423	Shop	Shoes & bags
620424	Shop	Gift/Souvenir
620425	Shop	Sporting Goods
620426	Shop	Toys
620427	Shop	Musical instruments
620500	Shop (food)	N
620501	Shop (food)	Bakery
620502	Shop (food)	Butcher

620503	Shop (food)	Convenience store
620504	Shop (food)	Delicatessen
620506	Shop (food)	Grocery store
620507	Shop (food)	Liquor/Wine/Beer shop
620508	Shop (food)	Specialty Food
630400	Other Tourist Attraction	N
630401	Other Tourist Attraction	Observatory
630500	Place of Worship	N
630501	Place of Worship	Church
630502	Place of Worship	Mosque
630503	Place of Worship	Synagogue

.POIMgr.CATCount:Integer; Property

The total number of categories (and sub-categories) in the POI database.

Delphi Example

```
procedure TForm1.Button731Click(Sender: TObject);
var i:integer;
begin
  // Set all categories invisible
  For i:=1 to MapPro1.POIMgr.CATCOunt do
  begin
    MapPro1.POIMgr.CATVisibility[i]:=false;
  end;
  // Set Pizzerias visible
  MapPro1.POIMgr.CATVisibility['Pizzeria']:=true;
  Mappro1.Redraw;
End;
```

.POIMgr.CATVisibility(Index:OLEVariant):Boolean; Property

Sets the visibility of the specified vategory. Index can either be the index # of the category in the categories array (not recommended, as this may cnahge as more categories are added), or the category name. Sub-categories may also be included at the end of the category name, separated by the paiping character.

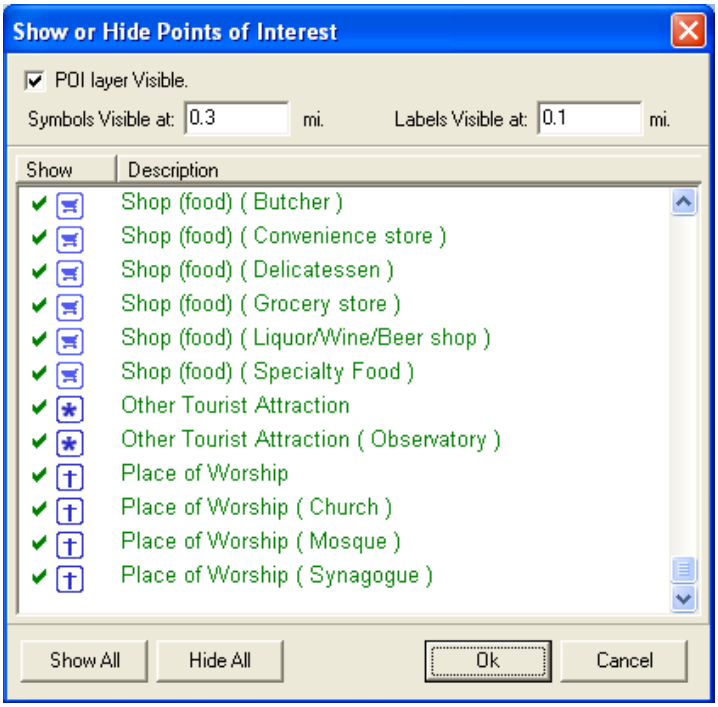
Delphi Example

```
procedure TForm1.Button731Click(Sender: TObject);
var i:integer;
begin
  // Set all categories invisible
  For i:=1 to MapPro1.POIMgr.CATCount do
  begin
    MapPro1.POIMgr.CATVisibility[i]:=false;
  end;
```

```
  // Set Pizzerias visible
  MapPro1.POIMgr.CATVisibility['Pizzeria']:=true;
  // the same could have been achieved by using
  // MapPro1.POIMgr.CATVisibility[560400]:=true;
  Mappro1.Redraw;
End;
```

.POIMgr.Dialog; Method

Opens up a dialog that allows the user to set the visibility of individual categories, set the limits of visibility for the icons and the text identifying POIS, etc.



Delphi Example

```
procedure TForm1.Button731Click(Sender: TObject);
var i:integer;
begin
  // Set the POI layer visibility to true
  For i:=1 to MapPro1.Visible:=true;
  // Open dialog to set visibility of individual categories
  Mappro1.POIMgr.dialog;
End;
```

.POIMgr.FormatHint:String Property

Defines the information that will be displayes when the mouse cursor rests on one of the POI icons on the map. Also see FormatLabel to see how to define the information that is used to label the POI location.

The format used for defining this property is as follows:

+	Is used to concatenate the various parts making up the string definition
@n	Specifies that the contents of the n-th field of the POI database should be inserted here. Note that the database field name could also be inserted here, but it is possible that field names may change in the future.
"…"	Double quotes enclose any literal text that should be added to the string
#nn	Two digit control code for things like CR, LF, etc.

Example:

```
POIMgr.FormatHint:='+"Name: "+@2+#13+#10+"Telephone:"+PHONE'
```

Would result in the following information being displayed, when the mouse pointer was over the POI icon, for North End Motor Sales.

Name:North End Motor Sales
Telephone:508-853-7665

The fileds that are accessible in the POI database are:

CAT_CODE – Category Code (See table at the beginning of this section)
STD_NAME – Name of the POI
PHONE – Telephone #
SIC_CODE1 – Industry Code
HOUSE_NUMBER – Street Address #
STREET – Street Name
CITY – City
COUNTY_CODE – County FIPS code
STATE_ABBR – State two char abbreviation
ZIP – 5 digit Zip Code
ZIP_4 – Zip+4

Delphi Example

```
procedure TForm1.Button71Click(Sender: TObject);
begin
  // This will turn the icons on at the 10 miles scale
  mappro1.poimgr.MilesMajor:=10;
  // This will turn on the labels at one mile scale
  mappro1.poimgr.MilesMinor:=1.5;
  // Does the *first* character in the format string have to be "+" ?
  mappro1.POIMgr.FormatLabel:='"Name:"+@2';
  // In the Hint format, we'll use a field name, as well - PHONE
  mappro1.POIMgr.FormatHint:='+"Name: "+@2+#13+#10+"Telephone:"+PHONE';
  Mappro1.POIMgr.Visible:=true;
end;
```

.POIMgr.FormatLabel:String Property

Defines the information that will be used to label the POI icons on the map. Also see FormatHint to see how to define the information that appears when the mouse cursor is on a POI icon.

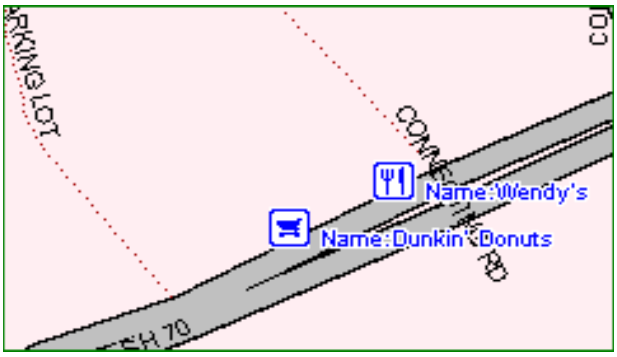
The format used for defining this property is as follows:

+	Is used to concatenate the various parts making up the string definition
@n	Specifies that the contents of the n-th field of the POI database should be inserted here. Note that the database field name could also be inserted here, but it is possible that field names may change in the future.
"…"	Double quotes enclose any literal text that should be added to the string

Example:

```
POIMgr.FormatLabel:='+"Name: "+@2'
```

Would result in the following labeling:



Delphi Example

```
procedure TForm1.Button71Click(Sender: TObject);
begin
  // This will turn the icons on at the 10 miles scale
  mappro1.poimgr.MilesMajor:=10;
  // This will turn on the labels at one mile scale
  mappro1.poimgr.MilesMinor:=1.5;
  // Does the *first* character in the format string have to be "+" ?
  mappro1.POIMgr.FormatLabel:='"Name:"+@2';
  // In the Hint format, we'll use a field name, as well - PHONE
  mappro1.POIMgr.FormatHint:='+"Name: "+@2+#13+#10+"Telephone:"+PHONE';
  Mappro1.POIMgr.Visible:=true;
end;
```

.POIMgr.GetCATCode(Name:WideString):Integer; Method

Returns the Category Code that corresponds to the specified name. A Sub-Category Code can also be obtained by appending the sub-category name, to the category name, separted by a piping character.

Delphi Example

```
procedure TForm1.Button71Click(Sender: TObject);
var i:integer;
    s:String;
begin
  //---
  // Get the code for the general category "Police Station"
  s:='Police Station';
  i:=Mappro1.POIMgr.GetCATCode(s);
  Listbox1.Items.add(s+' = '+ inttostr(i));

  // Get the code for the sub-category "Mexican Restaurant"
  s:='Restaurant|Mexican';
  i:=Mappro1.POIMgr.GetCATCode(s);
  Listbox1.Items.add(s+' = '+ inttostr(i));
end;
```

.POIMgr.GetCATName(CATcode:Integer):string; Method

Returns the Category Name that corresponds to the specified code. If the code is for a sub-Category, then it's also returned separated from the category name by a piping character.

Delphi Example

```
procedure TForm1.Button71Click(Sender: TObject);
var i:integer;
    s:String;
begin
  // Get the Category name for 570404
  i:=570404;
  s:=mappro1.POIMgr.GetCATName(i);
  Listbox1.Items.add(inttostr(i)+' = '+s);
end;
```

.POIMgr.MilesMajor:Double Property

Specifies the scale (miles) below which the POI icons will be visible on the map.

.POIMgr.MilesMinor:Double Property

Specifies the scale (miles) below which the POI icons will also be labeled.

.POIMgr.ReplaceBitmap(Index:OLEVariant; HBmp:Integer); Method

Replaces the icon for the POI specified by Index, with the user bitmap that the HBmp handle is referring to. The user bitmap needs to be of Windows BMP format. It can be either 12-bit or 24-bit, and 17x17 pixels. The color of the lower left corner pixel of the bitmap is treated as the transparent color.

Delphi Example

```
procedure TForm1.Button71Click(Sender: TObject);
var i:integer;
begin
  // -- Replace the icon used for Banks
  //    with user bitmap loaded in image object
  i:=image1.Picture.Bitmap.Handle;
  Mappro1.POIMgr.ReplaceBitmap('Bank',i);
  //  or Mappro1.POIMgr.ReplaceBitmap(12,i);
end;
```

.POIMgr.Visible:Boolean Property

Sets the visibility of the whole POI layer.

APPENDIX A - Overlay File Format

FORMAT OF OLDER PRECISION MAPPING OVERLAY FILE (OVR)

The format of the Precision Mapping older, binary format Overlay files is proprietary, and is only released to licensed users of the OCX unit, to be used according to the terms of their licensing Agreement with Undertow Software Corp.

(1) Each entity is in a complete block of data, each block of similar format.

(2) All record elements described here are required. Elements marked as reserved may be left blank, but still need to be allocated.

(3) File is terminated with a single FD byte.

```
Record Type (1)  - Line
---------------------------------------------------------------
Byte #    Variable         Type     Explanation
---------------------------------------------------------------
  1         CODE           Byte     1 = Line
  2..3      LAYER          Sm Int   Color, Style, etc. See (a)
  4         THICKNESS      byte     Line thickness ID
  5..10     Xc             Real     Start Pt. X coordinate
  11..16    Yc             Real     Start Pt. Y coordinate
  17..22    Xa             Real     End Pt. X coordinate
  23..28    Ya             Real     End Pt. Y coordinate
  29..34    Xb             Real     Reserved
  35..40    Yb             Real     Reserved


Example of Overlay file consisting of a single line
---------------------------------------------------
01 00 00 00 95 78 D5 40 00 C8 94 93 42 44 BE 17 95
4A C7 1B FD C7 94 40 BA 6B BC 17 84 10 6C 00 43 04
FE 00 C4 05 84 01 FD

Example of Overlay file consisting of a single line with a thick line attribute.
---------------------------------------------------
11 00 00 06 95 78 D5 40 00 C8 94 93 42 44 BE 17  95
4A C7 1B FD C7 94 40 BA 6B BC 17 84 10 6C 00  43 04
FE 00 C4 05 84 01 FD

Note (a)  When used, the LAYER variable holds the following information:

            bits   1..4  :  Foreground color
            bits   5..8  :  Background color
            bits   9..12 :  Fill Style
            bits  13..16 :  Line Style



Record Type (3)  - Circle (Defined in terms of a bounding box)
---------------------------------------------------------------
Byte #    Variable         Type     Explanation
---------------------------------------------------------------
  1         CODE           Byte     3 = Circle
  2..3      LAYER          Sm Int   Color, Style, etc. See (a)
  4         THICKNESS      byte     Line thickness ID
  5..10     Xc             Real     Top Left Pt. X coordinate
  11..16    Yc             Real     Top Left Pt. Y coordinate
  17..22    Xa             Real     Bot. Right Pt. X coordinate
  23..28    Ya             Real     Bot. Right Pt. Y coordinate
  29..34    Xb             Real     Reserved
  35..40    Yb             Real     Reserved

Example of Overlay file consisting of a Circle
---------------------------------------------------
```

```
03 00 00 01 95 50 15 38 00 C8 94 93 42 44 BE 17 95
FB 46 0A FD C7 94 93 42 A4 C4 17 23 04 F1 00 43 04
FE 00 C4 05 84 01 FD
```

Record Type (6) - Bezier

Byte #	Variable	Type	Explanation
1	CODE	Byte	6 = Bezier
2..3	LAYER	Sm Int	Color, Style, etc. See (a)
4	THICKNESS	byte	Line thickness ID
5..10	Xc	Real	Start Pt. X coordinate
11..16	Yc	Real	Start Pt. Y coordinate
17..22	Xa	Real	End Pt. X coordinate
23..28	Ya	Real	End Pt. Y coordinate
29..34	Xb	Real	MidPoint X coordinate
35..40	Yb	Real	MidPoint Y coordinate

Example of Overlay file consisting of a Bezier

```
06 00 00 01 95 50 15 38 00 C8 94 44 C2 32 BE 17 95
72 87 24 FD C7 94 8F 3A 7D BC 17 95 D6 0A E5 FD C7
94 D3 CF 34 C1 17 FD
```

Record Type (8) - Text

Byte #	Variable	Type	Explanation
1	CODE	Byte	8 = Text
2..3	LAYER	Sm Int	Color, Style, etc. See (a)
4	THICKNESS	byte	Line thickness ID
5..10	Xc	Real	Lower Left X coordinate
11..16	Yc	Real	Lower Left Y coordinate
17..22	Xa	Real	Reserved
23..28	Ya	Real	Reserved
29..34	Xb	Real	Reserved
35	L	Byte	Length of String
36..291	S	String	Text

Example of Overlay file consisting of Text

```
08 00 00 01 95 33 33 41 00 C8 94 00 80 41 BE 17 00
00 00 00 00 00 00 00 00 00 00 0F 00 00 00 00 00
06 53 41 4D 50 4C 45 37 22 A8 0C 37 22 AA 05 00 00
00 8C 10 AC 00 E1 24 DF 24 64 0D 37 22 EC 05 37 22
F7 FF FF FF 60 00 80 00 8C 10 94 00 00 00 5A 0D 30
C2 00 00 C0 C0 C0 02 00 00 00 00 00 FF 00 00 8C 10
7C 00 00 00 00 00 00 00 00 00 00 00 FD 12
```

Record Type (10) - Polyline

Byte #	Variable	Type	Explanation
1	CODE	Byte	10 = Polyline
2..3	LAYER	Sm Int	Color, Style, etc. See (a)
4	THICKNESS	byte	Line thickness ID
5..10	Xc	Real	Reserved
11..16	Yc	Real	Reserved
17..22	Xa	Real	Reserved
23..28	Ya	Real	Reserved
29..30	N	Sm Int	No. of points (max=100)
variable			N number of X,Y coordinate pairs.

Example of Overlay file consisting of a 3-segment Polyline.

```
0A 00 00 01 95 72 87 24 FD C7 94 BE CC 85 C0 17 95
78 D5 40 00 C8 94 52 B5 53 BB 17 04 00 95 78 D5 40
00 C8 94 93 42 44 BE 17 95 60 8C 3C FE C7 94 BE CC
85 C0 17 95 EB CF 05 FF C7 94 52 B5 53 BB 17 95 72
```

```
87 24 FD C7 94 40 BA 6B BC 17 FD
```

Record Type (12) - Bubble

Byte #	Variable	Type	Explanation
1	CODE	Byte	12 = Bubble
2..3	LAYER	Sm Int	Color, Style, etc. See (a)
4	TP	byte	Reserved
5..10	Xc	Real	Bubble X coordinate
11..16	Yc	Real	Bubble Y coordinate
17..22	Xa	Real	Reserved
23..28	Ya	Real	Reserved
29..284	S	string[256]	Bubble string. First byte is the actual length of the string

Example of Overlay file consisting of a text bubble

```
0C 00 00 01 95 50 15 38 00 C8 94 93 42 44 BE 17 95
E7 05 CD FC C7 94 85 4F 23 C1 17 0D 42 75 62 62 6C
65 20 53 61 6D 70 6C 65 35 70 0C 7F 35 AA 05 00 00
73 14 1D EC 00 E1 24 DF 24 58 18 7F 35 EC 05 37 22
F7 FF FF FF 60 00 52 4E 14 1D D4 00 27 25 DF 24 58
18 7F 35 B4 18 7F 35 14 1D C4 00 00 00 8E 0D 30 C2
00 00 C0 C0 C0 02 00 00 00 00 00 29 7C 2A 14 1D AC
00 00 00 00 00 00 00 00 00 00 00 14 00 14 00 01 00
14 1D 98 00 00 00 00 00 00 00 00 00 01 00 14 1D 7F
35 F8 18 7F 35 CE 00 14 1D 80 00 00 00 00 2C 19 7F
35 44 19 7F 35 58 19 7F 35 00 00 00 00 C6 00 88 00
A3 40 DF 24 E0 18 7F 35 7F 59 DF 24 E0 18 7F 35 E0
18 7F 35 AA 05 00 00 01 14 1D 4C 00 E1 24 DF 24 F8
18 7F 35 EC 05 37 22 FF FF FF 00 60 00 AB 01 68 19
14 00 04 25 DF 24 F8 18 7F 35 A4 06 37 22 04 02 03
02 14 1D 20 00 27 25 DF 24 F8 18 7F 35 14 10 7F 35
14 1D 10 00 14 1D 0C 00 14 1D 08 00 FD
```

Record Type (14) - Group

Byte #	Variable	Type	Explanation
1	CODE	Byte	14 = Group
2..3	LAYER	Sm Int	Reserved
4	THICKNESS	byte	Reserved
5..10	Xc	Real	Reserved
11..16	Yc	Real	Reserved
17..22	Xa	Real	Reserved
23..28	Ya	Real	Reserved
29..30	QCOUNT	Sm Int	No. of elements (max=100)
variable			QCOUNT number of records, each record consistent with the format given above.

Example of Overlay file consisting of a a Grouped object made up of a line, a circle and a polyline

```
1E 10 9C 02 49 F8 B0 00 52 F8 B0 00 58 F8 B0 00 5A
F8 B1 00 5B F8 B1 00 60 F8 B1 00 03 00 01 00 00 01
95 33 33 41 00 C8 94 00 80 41 BE 17 95 CD 0C 24 FD
C7 94 9A 99 73 BC 17 0A 37 22 AA 05 00 00 02 5C 0B
0C 00 0A 00 00 01 95 CD 0C 24 FD C7 94 00 00 75 C1
17 95 00 C0 39 00 C8 94 33 B3 3A BB 17 04 00 95 00
C0 39 00 C8 94 00 80 41 BE 17 95 66 66 F9 FE C7 94
00 00 75 C1 17 95 33 B3 CC FE C7 94 33 B3 3A BB 17
95 CD 0C 24 FD C7 94 00 80 82 BC 17 03 00 00 01 95
CD 4C 32 00 C8 94 00 80 41 BE 17 95 33 F3 32 FD C7
94 00 80 41 C4 17 30 C6 00 00 C0 C0 C0 00 00 00 00
00 FD
```

Record Type (FD) - End of file

Byte #	Variable	Type	Explanation
1	CODE	Byte	FD = End Of File

APPENDIX B – Color Palette

COLOR PALETTE

The color palette used in the OCX unit(s) is a 256 color palette. The palette uses the 20 system colors, plus 16 colors defined for shading, with the rest of the colors in the palette being undefined.

The 16 colors used for shading are as follows (in RGB hex notation):

1 $DFDFDF
2 $BAC9BC
3 $B4CFCA
4 $D1DAD5
5 $CACABB
6 $BBC8C8
7 $C5BEC5
8 $DDCECE
9 $B5CECE
10 $C5BEBE
11 $A0B8C5
12 $ABCCCF
13 $B8BECB
14 $C7C6BC
15 $B3C2C6
16 $C9E2E2

The user should be aware that the entries in the [Shades] section of the .CFG file may be of two different types. If the high bit is set, then that entry is simply an index to the above table and not a real color. If the highbit is not set, then the last three bytes are the RGB color (remember they are reversed). For example, an entry of STSH=17,$08000003 indicates that the 17th state (IL) uses the 3rd color in the above table, whereas an entry of STSH=17,$000000FF would indicate that the 17th state uses the color red.

It is recommended that if any sort of bitmap is used with the OCX, these 16 colors be part of its palette to avoid any palette conflict problems.

APPENDIX C – Configuration File

CONFIGURATION (.CFG) FILE

This Appendix contains a copy of a 'typical' .CFG file that is created every time an application using the OCX terminates normally, or when the Map object properties change at design time. It is read every time the application is started. Note that the information written out to the .CFG file is for general reference only.

```
[Directories]
CDSTATE=D:\pmap40\states
CDDATA1=D:\pmap40\data1
CDDATA2=D:\pmap40\data2
CDDATA3=D:\pmap40\data3
CDOVR=
CDLIB=
CDCONT=
CDHELP=
BMP=
OVR=
LIB=

[Registration]
NAME= {Reserved}
IDTYPE= {Reserved}

[Visible]
UNDER=TRUE {Underlay Visibility flag}
OVER=TRUE {Overlay Visibility flag}
MAIN=TRUE {Mainlay Visibility flag}
DBASE=TRUE

[Modules]
ASP=1.000
BRATIO=4.000
TIGER=2
CSPC=5
PMODE=1
BKPNT=FALSE

[Overlays]
LMAIN=TRUE   {main}
LHYDRO=TRUE  {Hydro}
LHNET=TRUE   {Highway Network}
LSTATE=TRUE  {State Outlines}
LCITY=TRUE   {City Outlines}
LAIR=TRUE    {Airports}
LLAND=TRUE   {Landmarks}
LCNTY=TRUE   {County Outlines}

[Options]
HINT=TRUE
TOOL=TRUE
STATUS=TRUE
GLOBE=TRUE
XDT=TRUE
PC=TRUE
```

```
BEEP=TRUE
OPTID=TRUE
UMONO=0
UMNT=10

[ Font Info ]
IMAX=9
IMIN=6
SMJ=0.500
SHADO=TRUE
SHFACT=128
SHPOL=128
BWON=FALSE
SCBAR=TRUE
LVIEW=TRUE
LWIND=-133.706624,16.801664,-55.718080,58.773184

[Printer]
MLT=0.25 {Left margin}
MRT=0.25 {Right margin}
MTP=0.25 {Top margin}
MBT=0.25 {Bottom margin}
SIZE=1
FSIZE=1
TITLE=Current Map
FPS=0
XYMODE=0
UNIT=1
PRTBDR=2

[Search]
PLC=LEMONT  {Last city/place searched for}
STR=127  {Last street searched for}
ZIP=60439 {Last Zip Code searched for}
ACODE=630 {Last Area Code searched for}
STNUM=56 {Last block/Street number searched for}
SMODE=2
ZMARD=1,0,2
ZMARD=2,0,0
ZMARD=3,0,0
ZMARD=4,0,0
ZMARD=5,0,0
ZMARD=6,0,2

[Landmarks] {Visibily of each 1:Visible, 0:Invisible}
LAND=1,1
LAND=2,1
LAND=3,1
LAND=4,1
LAND=5,1
LAND=6,1
LAND=7,1
LAND=8,1
LAND=9,1
LAND=10,1
LAND=11,1
LAND=12,1
LAND=13,1
LAND=14,1
LAND=15,1
```

```
LAND=16,1
LAND=17,1
LAND=18,1
LAND=19,1
LAND=20,1
LAND=21,1
LAND=22,1
LAND=23,1
LAND=24,1
LAND=25,1
LAND=26,1
LAND=27,1
LAND=28,1
LAND=29,1
LAND=30,1
LAND=31,1
LAND=32,1
LAND=33,1
LAND=34,1
LAND=35,1
LAND=36,1
LAND=37,1
LAND=38,1
LAND=39,1
LAND=40,1
LAND=41,1
LAND=42,1
LAND=43,1
LAND=44,1
LAND=45,1
LAND=46,1
LAND=47,1
LAND=48,1
LAND=49,1
LAND=50,1
LAND=51,1
LAND=52,1
LAND=53,1
LAND=54,1
LAND=55,1
LAND=56,1
LAND=57,1
LAND=58,1
LAND=59,1
LAND=60,1
LAND=61,1
LAND=62,1
LAND=63,1
LAND=64,1
LAND=65,1
LAND=66,1
LAND=67,1
LAND=68,1
LAND=69,1

[Layers]
The visibility flags for Layers 1-29 are stored in this section of
the .CFG file.  A flag of 1 indicates visible, while a flag of "0"
indicates invisible.  The 29 layers used are composite layers
based primarily on Tiger type entities, as identified below.  Note
```

that the composition of each layer may be changed at any time, if required.

Composition of the 29 layers, in terms of their Tiger identifiers. Consult the Tiger documentation, of the Precision Mapping Streets help file for the Tiger classification IDs.

```
Layer # 1   =  Reserved
Layer # 2   =  Reserved
Layer # 3   =  Reserved


Layer # 4   =  Reserved
Layer # 5   =  Reserved
Layer # 6   =  Reserved
Layer # 7   =  Reserved
Layer # 8   =  Reserved
Layer # 9   =  A10, A11, A12, A13, A14, A15, A16,
               A17, A18
Layer # 10  =  A20, A21, A22, A23, A24, A25,
               A26, A27, A28
Layer # 11  =  A30, A31, A32, A33, A34, A35
Layer # 12  =  A36, A37, A38
Layer # 13  =  A00, A01, A02, A03, A04, A05, A06,
               A07, A08, A45, A46, A47, A48,
               A50,A51, A52, A53, A60, A61, A62,
               A63, A64, A65, A70, A71, A72, A73
Layer # 14  =  A40, A41, A42, A43, A44
Layer # 15  =  Reserved
Layer # 16  =  Reserved
Layer # 17  =  B00, B01, B02, B03, B10, B11, B12,
               B13, B20, B21, B22, B23, B30, B31,
               B32, B33, B40, B50, B51, B52
Layer # 18  =  C00, C10, C20, C30, C31
Layer # 19  =  D00, D10, D20, D21, D22, D23, D24,
               D25, D26, D27, D28, D29, D30, D31,
               D32, D33, D34, D35, D36, D37, D40,
               D41, D42, D43, D44, D50, D51, D52,
               D53, D54, D55, D60, D61, D62, D63,
               D64, D65, D66, D70, D71, D80, D81,
               D82, D83, D84, D85, D90, D91
Layer # 20  =  E00, E10, E20, E21, E22
Layer # 21  =  F00, F10, F11, F12, F13, F14, F15,
               F20, F21, F22, F23, F24, F25, F30,
               F40, F50, F60, F70, F71, F72, F73,
               F74, F80, F81, F82
Layer # 22  =  Reserved
Layer # 23  =  H00, H01, H10, H11, H13, H20, H21,
               H30, H31, H40, H41, H50 ,H51, H53,
               H60, H70, H71, H73, H74, H75, H80,
               H81
Layer # 24  =  H02, H12, H22, H32, H42, H72
Layer # 25  =  Reserved
Layer # 26  =  X00
Layer # 27  =  Reserved
Layer # 28  =  Reserved
Layer # 29  =  Reserved
```

```
LAYER=1,1
LAYER=2,1
LAYER=3,1
LAYER=4,1
LAYER=5,1
LAYER=6,1
LAYER=7,1
LAYER=8,1
LAYER=9,1
LAYER=10,1
LAYER=11,1
LAYER=12,1
LAYER=13,1
LAYER=14,1
LAYER=15,1
LAYER=16,1
LAYER=17,1
LAYER=18,1
LAYER=19,1
LAYER=20,1
LAYER=21,1
LAYER=22,1
LAYER=23,1
LAYER=24,1
LAYER=25,1
LAYER=26,1
LAYER=27,1
LAYER=28,1
LAYER=29,1

[Shades]
CNTYON=TRUE  {Shade Counties}
MCDON=TRUE   {Shade MCDs}
PLCON=TRUE   {Shade Places}
AUTOQ=TRUE   {AutoQuery flag}
STATEON=TRUE  {Shade States}

[Fills]
FILLS=
FILLS=
FILLS=

[Shades]
STSH=1,$08000004
STSH=2,$08000000
STSH=3,$08000000
STSH=4,$08000002
STSH=5,$08000003
STSH=6,$08000003
STSH=7,$08000000
STSH=8,$08000004
STSH=9,$08000006
STSH=10,$08000002
STSH=11,$08000003
STSH=12,$08000002
STSH=13,$08000003
STSH=14,$08000000
STSH=15,$08000000
STSH=16,$08000003
```

```
STSH=17,$08000006
STSH=18,$08000002
STSH=19,$08000002
STSH=20,$08000006
STSH=21,$08000003
STSH=22,$08000002
STSH=23,$08000007
STSH=24,$08000003
STSH=25,$08000003
STSH=26,$08000007
STSH=27,$08000006
STSH=28,$08000006
STSH=29,$08000004
STSH=30,$08000006
STSH=31,$08000003
STSH=32,$08000004
STSH=33,$08000006
STSH=34,$08000006
STSH=35,$08000003
STSH=36,$08000004
STSH=37,$08000003
STSH=38,$08000003
STSH=39,$08000004
STSH=40,$08000002
STSH=41,$08000002
STSH=42,$08000007
STSH=43,$08000000
STSH=44,$08000004
STSH=45,$08000006
STSH=46,$08000004
STSH=47,$08000002
STSH=48,$08000006
STSH=49,$08000006
STSH=50,$08000002
STSH=51,$08000004
STSH=52,$08000000
STSH=53,$08000004
STSH=54,$08000006
STSH=55,$08000003
STSH=56,$08000002
STSH=57,$00DFDFDF
STSH=58,$00B5CECE
STSH=59,$00BAC9BC
STSH=60,$00C5BEBE
STSH=61,$00B4CFCA
STSH=62,$00A0B8C5
STSH=63,$00D1DAD5
STSH=64,$00ABCCCF
STSH=65,$00CACABB
STSH=66,$00B8BECB
STSH=67,$00BBC8C8
STSH=68,$00C7C6BC
STSH=69,$00C5BEC5
STSH=70,$00B3C2C6
STSH=71,$00DDCECE
STSH=72,$00C9E2E2

[Route]
PRIOR=1,65,5
PRIOR=2,50,4
```

```
PRIOR=3,45,3
PRIOR=4,35,3
PRIOR=5,5,1
HTYPE=2
HCLR=65535
TFACTOR=5
RTERAD=10
```

APPENDIX D – Abbreviations in Searching

ABBREVIATIONS IN SEARCHING

When searching for a street, the following abbreviations may be used:

Suffix Abbreviations
Lane, Ln
Street, St
Place, Pl
Drive, Dr
Trail, Tr
Avenue, Ave
Parkway, Pkwy
Circle, Cir
Highway, Hiway, Hwy
Terrace, Ter
Court, Ct
Turnpike, Tpke
Road, Rd

For example, Searching for 'Linda Dr' or 'Linda Drive' should return the same results.

Prefix Abbreviations
West, W
East, E
North, No., N
South, So., S

For example, searching for 'West Main', or 'W Main' should return the same results.

APPENDIX E - Visual Basic Sample Code

Visual Basic Sample Code
This Appendix contains sample Visual Basic code using various aspects of MapPro.OCX.

General Basic Examples

Note: Declarations, Events and Functions have been included ' to illustrate how MapPro can be used with Visual Basic.

```
'The following declarations are required in order to use the
'WinApi calls provided in the examples.

Public Type POINTAPI
        x As Long
        y As Long
End Type

Dim LPoint As POINTAPI
Declare Function MoveToEx Lib "gdi32" (ByVal hdc As Long, ByVal x As Long, ByVal
y As Long, LpPoint As POINTAPI) As Long
Declare Function LineTo Lib "gdi32" (ByVal hdc As Long, ByVal x As Long, ByVal y
As Long) As Long
Declare Function CreatePen Lib "gdi32" (ByVal nPenStyle As Long, ByVal nWidth As
Long, ByVal crColor As Long) As Long
Declare Function SelectObject Lib "gdi32" (ByVal hdc As Long, ByVal hObject As
Long) As Long
Declare Function DeleteObject Lib "gdi32" (ByVal hObject As Long) As Long
Declare Function Ellipse Lib "gdi32" (ByVal hdc As Long, ByVal X1 As Long, ByVal
Y1 As Long, ByVal X2 As Long, ByVal Y2 As Long) As Long
Declare Function SetROP2 Lib "gdi32" (ByVal hdc As Long, ByVal nDrawMode As
Long) As Long
Declare Function GetDC Lib "user32" (ByVal hwnd As Long) As Long
Declare Function GetDeviceCaps Lib "gdi32" (ByVal hdc As Long, ByVal nIndex As
Long) As Long
Declare Function ReleaseDC Lib "user32" (ByVal hwnd As Long, ByVal hdc As Long)
As Long

' The following functions can be used to parse delimited strings

Public Function CommaDelStr(ByRef CommaStr) As String
 ' If CommaStr contains a comma delimited string
 ' Then CommaDelStr returns all characters up to
 ' but not including the first comma.  All Characters
 ' up to (and including) the first comma are removed
 ' from CommaStr
Dim Tstr, CommaPos
   Tstr = CommaStr
   CommaPos = InStr(1, Tstr, ",", vbTextCompare)

   If CommaPos = 0 Then
    CommaDelStr = CommaStr ' Return the entire string since it has no comma
   ElseIf CommaPos > 0 Then
    CommaDelStr = Left(Tstr, CommaPos - 1)
    CommaStr = Right(Tstr, Len(Tstr) - CommaPos)
```

```
   End If
End Function


Public Function TabDelStr(ByRef TabStr) As String
 ' If TabStr contains a Tab delimited string
 ' Then TabDelStr returns all characters up to
 ' but not including the first Tab.  All Characters
 ' up to (and including) the first Tab are removed
 ' from TabStr

Dim Tstr, TabPos
   Tstr = TabStr
   TabPos = InStr(1, Tstr, Chr(9), vbTextCompare)
   If TabPos = 0 Then
    TabDelStr = TabStr  ' Return the entire string since it has no comma
   ElseIf TabPos > 0 Then
    TabDelStr = Left(Tstr, TabPos - 1)
    TabStr = Right(Tstr, Len(Tstr) - TabPos)
   End If
End Function


Private Sub MapPro1_Find()
 ' The Find Event is called for every street segment located by
 ' the MapPro1.FindStr Function.
 ' The TabDelStr Function shown is not included as an OCX method.
 ' See the examples for parsing comma and tab delimited strings.
 ' TabDelStr returns all text up to the first Tab.  The function
 ' removes all characters up to and including the tab from Tstr.
 ' This example converts the Tab Delimited String to a Comma Delimited
 ' and places it in a list.
 ' (Remember to clear the list before calling FindStr)

Dim Tstr, ListStr
   Tstr = MapPro1.Street  ' Put the street results in a temp var
   ListStr = TabDelStr(Tstr)                  ' Street Name
   ListStr = ListStr & "," & TabDelStr(Tstr) ' City
   ListStr = ListStr & "," & TabDelStr(Tstr) ' Address range
   ListStr = ListStr & "," & TabDelStr(Tstr) ' State
   ListStr = ListStr & "," & TabDelStr(Tstr) ' Zip
   ListStr = ListStr & "," & TabDelStr(Tstr) ' Longitude
   ListStr = ListStr & "," & TabDelStr(Tstr) ' Latitude
   ListStr = ListStr & "," & TabDelStr(Tstr) ' Distance
   List1.AddItem (ListStr)
End Sub


Private Sub MapPro1_paintAfter(ByVal DC As Long)
 ' This event is where much of your application drawing activity
 ' can be handled.  Lines will be drawn over streets.
 ' This example shows several WinApi calls

Dim Lp As POINTAPI
Dim hp As Long
Dim tmpXLoc, tmpYLoc, diff As Double
   Call MapPro1.DrawScalebar(DC, 1, 10) ' Place the scale bar on the map
   MagEd.Text = MapPro1.Magnitude       ' Report some properties to user
   ScaleEd.Text = MapPro1.Scale
   MileEd.Text = MapPro1.Miles
```

```
   AspEd.Text = MapPro1.Screen_Aspect

   hp = CreatePen(0, 1, 0)            ' Pen used to draw with WinApi
   hp = SelectObject(DC, hp)

   With MapPro1
      tmpXLoc = (.LonLeft + .LonRight) / 2  ' Get the center of the map
      tmpYLoc = (.LatTop + .LatBottom) / 2
      .DeleteAllItems                       ' Clear any previous items.
      .SetItem 1, tmpXLoc, tmpYLoc          ' Place an item on the map
      .SetItemBitmap 1, Picture1.Picture.Handle ' Associate bitmap with item
      .SetItemString 1, "My Bitmap"         ' String will appear above bitmap
      '  Draw a text bubble pointing to item.
      Call .DrawBubble(DC, tmpXLoc, tmpYLoc, "Bubble")

      ' Draw line from lower left to upper right using WinApi calls
      Call MoveToEx(DC, .Lon2Int(.LonLeft), .Lat2Int(.LatBottom), Lp)
      Call LineTo(DC, .Lon2Int(.LonRight), .Lat2Int(.LatTop))
   End With
   Call DeleteObject(SelectObject(DC, hp)) ' Do NOT forget this!!!!!
End Sub


Private Sub MapPro1_PaintBefore(ByVal dc As Long)
' Lines can be drawn under streets in this event if all map
' shading is turned OFF.  The line will disappear if shading
' is on however.
' The DirectBefore event can be used in a similar manner when
' when using DirectDraw to draw to another bitmap or the printer.
Dim tmpXLoc, tmpYLoc As Double

   With MapPro1
      tmpXLoc = (.LonLeft + .LonRight) / 2  ' Get the center of the map
      tmpYLoc = (.LatTop + .LatBottom) / 2
      '  Draw a red line from upper right to center of the map.
      Call .DrawLine(DC,.LonRight,.LatTop,tmpXLoc,tmpYLoc,1,ClRed,13);
   End With
End Sub



Private Sub DirDrawNotScaled_Click()
' Demonstrates DirectDraw not Scaled.
' DirDrawFrm contains  a PictureBox called DirDrawPic
' The Map aspect ratio is maintained to prevent
' distortion.

   DirDrawFrm.Show
   DirDrawFrm.Caption = "Direct Draw Not Scaled "
   DirDrawFrm.DirDrawPic.Width = MapPro1.Width \ 4
   DirDrawFrm.DirDrawPic.Height = MapPro1.Height \ 4
   With DirDrawFrm
      Call MapPro1.DirectDraw(.DirDrawPic.hdc, 0, 0, .DirDrawPic.Width,
.DirDrawPic.Height, True,
                        False, True)
   End With
End Sub


Private Sub DirdrawScaled_Click()
```

```
' Demonstrates DirectDraw Scaled.
' DirDrawFrm contains  a PictureBox called DirDrawPic
' The Map aspect ratio is maintained to prevent
' distortion.

   DirDrawFrm.Show
   DirDrawFrm.Caption = "Direct Draw Scaled "
   DirDrawFrm.DirDrawPic.Width = MapPro1.Width \ 4
   DirDrawFrm.DirDrawPic.Height = MapPro1.Height \ 4
   With DirDrawFrm
      Call MapPro1.DirectDraw(.DirDrawPic.hdc, 0, 0, .DirDrawPic.Width,
.DirDrawPic.Height, True,
                        True, False)
End With
End Sub



Private Sub GotoBtn_Click()
Dim Lon, Lat

   Lon = -83.436
   Lat = 42.732
   Call MapPro1.GotoPoint(Lon, Lat)
End Sub


Private Sub ZoomWin_Click()
Dim X1, X2, Y1, Y2 As Variant

   X1 = -88
   Y1 = 41.56
   X2 = -88.01
   Y2 = 41.575
   Call MapPro1.Zoomwindow(X1, Y1, X2, Y2)
End Sub
```

APPENDIX F – CFCC Definitions

Census Feature Class Codes (CFCC) Definitions

A CFCC is used to identify the most noticeable characteristic of a feature. The CFCC is applied only once to a chain or landmark with preference given to classifications that cover features that are visible to an observer and are part of the ground transportation network. Thus a road that is also the boundary of a town would have a CFCC describing its road characteristics not its boundary characteristics. The CFCC, as used in the TIGER/Line(TM) files, is a three-character code; the first character is a letter describing the feature class; the second character is a number describing the major category; and the third character is a number describing the minor category.

```
A - Roads
B - Railroad - Ground Transportation
D - Landmarks
E - Physical Features
F - Non Visible Features
H - Hydrography
X - NonClassified
```

Feature Class A, Road

The definition of a divided highway has been the source of considerable discussion. Earlier specifications have defined a "divided" road as having "... opposing traffic lanes that are physically separated by a median strip no less than 70 feet wide in former GBF/DIME areas or no less than 200 feet wide in nonGBF/DIME areas." This definition caused confusion in the proper coding of interstates having narrow medians. To clarify the situation, the Census Bureau now uses the term "divided" to refer to a road with opposing traffic lanes separated by any size median, and "separated" to refer to lanes that are represented in the Census TIGER data base as two distinct complete chains.

Earlier operations may have depicted widely separated lanes as a single line in the data base or created separate lines when the median was small, depending on the available source used during the update. The term "rail line in center" indicates that a rail line shares the road right-of-way. The rail line may follow the center of the road or be directly next to the road, representation is dependent upon the available source used during the update. The rail line can represent a railroad, a street car line, or other carline.

Road With Major Category Unknown [A00-A08]

Source materials do not allow determination of the major road category. These codes should not, under most circumstances, be used since the source materials usually provide enough information to determine the major category.

```
A00 Road, major and minor categories unknown
A01 Road, unseparated
A02 Road, unseparated, in tunnel
A03 Road, unseparated, underpassing
A04 Road, unseparated, with rail line in center
A05 Road, separated
A06 Road, separated, in tunnel
A07 Road, separated, underpassing
A08 Road, separated, with rail line in centercategory
```

Primary Highway with Limited Access [A10-A18]

This road is distinguished by the presence of interchanges, access to the highway is by way of ramps, and there are multiple lanes of traffic. A road in this category has the opposing traffic

lanes "divided" by a median strip. Interstate highways and some toll highways are in this major category. The TIGER/Line(TM) files may depict the opposing lanes of a road in this category as two distinct lines; in this case the road is called "separated."

A10 Primary road with limited access or interstate highway, major category used alone when the minor category could not be determined

A11 Primary road with limited access or interstate highway, unseparated

A12 Primary road with limited access or interstate highway, unseparated, in tunnel

A13 Primary road with limited access or interstate highway, unseparated, underpassing

A14 Primary road with limited access or interstate highway, unseparated, with rail line in center

A15 Primary road with limited access or interstate highway, separated

A16 Primary road with limited access or interstate highway, separated, in tunnel

A17 Primary road with limited access or interstate highway, separated, underpassing

A18 Primary road with limited access or interstate highway, separated, with rail line in center

Primary Road without Limited Access [A20-A28]

A road in this major category must be hard surface, that is, concrete or asphalt, and may be divided or undivided and have multi-lane or single lane characteristics. This road has intersections with other roads, usually controlled with traffic lights. This major category includes nationally and regionally important highways that do not have limited access as required by major category A1. Thus, major category A2 includes most U.S. and State highways and some county highways that connect cities and larger towns

A20 Primary road without limited access, U.S. and State highway, major category used alone when the minor category could not be determined

A21 Primary road without limited access, U.S. and State highways, unseparated

A22 Primary road without limited access, U.S. and State highways, unseparated, in tunnel

A23 Primary road without limited access, U.S. and State highways, unseparated, underpassing

A24 Primary road without limited access, U.S. and State highways, unseparated, with rail line in center

A25 Primary road without limited access, U.S. and State highways, separated

A26 Primary road without limited access, U.S. and State highways, separated, in tunnel

A27 Primary road without limited access, U.S. and State highways, separated, underpassing in center

A28 Primary road without limited access, U.S. and State highways, separated, with rail line

Secondary and Connecting Road [A30-A38]

A road in this major category must be hard surface, that is, concrete or asphalt, usually undivided with single lane characteristics. This road has intersections with other roads, controlled with traffic lights and stop signs. This major category includes State and county highways that connect smaller towns, subdivisions, and neighborhoods, thus the road is smaller than a road in major category A2. This road, usually with a local name along with a route number, intersects with many other roads and driveways.

A30 Secondary and connecting road, State and county highways, major category used alone when the minor category could not be determined

A31 Secondary and connecting road, State and county highways, unseparated

A32 Secondary and connecting road, State and county highways, unseparated, in tunnel

A33 Secondary and connecting road, State and county highways, unseparated, underpassing

A34 Secondary and connecting road, State and county highways, unseparated, with rail line in center

A35 Secondary and connecting road, State and county highways, separated

A36 Secondary and connecting road, State and county highways, separated, in tunnel

A37 Secondary and connecting road, State and county highways, separated, underpassing

A38 Secondary and connecting road, State and county highway, separated, with rail line in center

Local, Neighborhood, and Rural Road [A40-A48]

A road in this major category is used for local traffic, usually with a single lane of traffic in each direction. In an urban area, this is a neighborhood road and street that is not a thoroughfare belonging in categories A2 or A3. In a rural area, this is a short distance road connecting the smallest towns; the road may or may not have a State or county route number. In addition, this major category includes scenic park roads, unimproved or unpaved roads, and industrial roads. Most roads in the Nation are classified in this major category.

A40 Local, neighborhood, and rural road, city street, major category used alone when the minor category could not be determined

A41 Local, neighborhood, and rural road, city street, unseparated

A42 Local, neighborhood, and rural road, city street, unseparated, in tunnel

A43 Local, neighborhood, and rural road, city street, unseparated, underpassing

A44 Local, neighborhood, and rural road, city street, unseparated, with rail line in center

A45 Local, neighborhood, and rural road, city street, separated

A46 Local, neighborhood, and rural road, city street, separated, in tunnel

A47 Local, neighborhood, and rural road, city street, separated, underpassing

A48 Local, neighborhood, and rural road, city street, separated, with rail line in center

Vehicular Trail [A50-A53]

A road in this major category is usable only by four-wheel drive vehicles and is usually a one lane, dirt trail. The road is found almost exclusively in a very rural area, sometimes the road is called a fire road or logging road and may include an abandoned railroad grade where the tracks have been removed. Minor, unpaved roads usable by ordinary cars and trucks belong in major category A4.

A50 Vehicular trail, road passable only by four-wheel drive (4WD) vehicle, major category used alone when the minor category could not be determined

A51 Vehicular trail, road passable only by 4WD vehicle, unseparated

A52 Vehicular trail, road passable only by 4WD vehicle, unseparated, in tunnel

A53 Vehicular trail, road passable only by 4WD vehicle, unseparated, underpassing

Road with Special Characteristics [A60-A65]

A road, portion of a road, intersection of a road, or the ends of a road that are parts of the vehicular highway system that have separately identifiable characteristics

A60 Road with characteristic unspecified, major category used alone when the minor category could not be determined

A61 Cul-de-sac, the closed end of a road that forms a loop or turn around (the node symbol that appears on some census maps is not included in the TIGER/Line(TM) files)

A62 Traffic circle, the portion of a road or intersection of roads that form a roundabout (the node symbol that appears on some census maps is not included in the TIGER/Line(TM) files)

A63 Access ramp, the portion of a road that forms acloverleaf or limited access interchange (the node symbol that appears on some census maps is not included in the TIGER/Line(TM) files)

A64 Service drive, the road or portion of a road thatprovides access to businesses, facilities, and rest areas along a limited access highway, this frontage road may intersect other roads and be named

A65 Ferry crossing, the portion of a road over water that consists of ships, carrying automobiles, connecting roads on opposite shores

Road as Other Thoroughfare [A70-A73]

A road that is not part of the vehicular highway system. This road is used by bicyclists or pedestrians and is typically inaccessible to mainstream motor traffic except by service vehicles. A stair and walkway may follow a road right-of-way and be named as if it were a road. This major category includes foot and hiking trails located on park and forest land.

A70 Other thoroughfare, major category used alone when the minor category could not be determined

A71 Walkway, nearly level road for pedestrians, usually unnamed

A72 Stairway, stepped road for pedestrians, usually unnamed

A73 Alley, road for service vehicles, usually unnamed,located at the rear of buildings and property

Feature Class B, RAILROAD & Ground Transportation

Railroad With Major Category Unknown [B00-B03]

Source materials do not allow determination of the major railroad category. These codes should not, under most circumstances, be used since the source materials usually provide enough information to determine the major category.

B00 Railroad, major and minor categories unknown

B01 Railroad track, not in tunnel or underpassing, major category used alone when the minor category could not be determined

B02 Railroad track, in tunnel

B03 Railroad track, underpassing

Railroad Main Line [B10-B13]

A railroad in this major category is the primary track that provides service between destinations. A main line track often carries the name of the owning and operating railroad company.

```
B10 Railroad main track, major category used alone when the minor category could not be
determined

B11 Railroad main track, not in tunnel or underpassing

B12 Railroad main track, in tunnel

B13 Railroad main track, underpassing
```

Railroad Spur [B20-B23]

A railroad in this major category is the track that leaves the main track, ending in an industrial park, factory, or warehouse area or forming a siding along the main track.

```
B20 Railroad spur track, major category used alone when the minor category could not be
determined

B21 Railroad spur track, not in tunnel or underpassing

B22 Railroad spur track, in tunnel

B23 Railroad spur track, underpassing
```

Railroad Yard [B30-B33]

A railroad yard track has parallel tracks that form a working area for the railroad company. Train cars and engines are repaired, switched, and dispatched from a yard.

```
B30 Railroad yard track, major category used alone when the minor category could not be
determined

B31 Railroad yard track, not in tunnel or underpassing

B32 Railroad yard track, in tunnel

B33 Railroad yard track, underpassing
```

Railroad with Special Characteristics [B40]

A railroad or portions of a railroad track that are parts of the railroad system and have separately identifiable characteristics.

```
B40 Railroad ferry crossing, the portion of a railroad over water that consists of ships,
carrying train cars to connecting railroads on opposite shores.  These are primarily located
on the Great Lakes.
```

Railroad as Other Thoroughfare [B50-B52]

A railroad that is not part of the railroad system. This major category is for a specialized rail line or railway that is typically inaccessible to mainstream railroad traffic.

```
B50 Other rail line, major category used alone when the minor category could not be
determined

B51 Carline, a track for street cars, trolleys, and other mass transit rail systems, used
when the carline is not part of the road right-of-way
```

B52 Cog railroad, incline railway, or logging tram

Feature Class C, Miscellaneous Ground Transportation

Miscellaneous Ground Transportation With Category Unknown [C00]

Source materials do not allow determination of the miscellaneous ground transportation category. This code should not, under most circumstances, be used since the source materials usually provide enough information to determine the major category.

C00 Miscellaneous ground transportation, not road or railroad, major and minor categories unknown

Pipeline [C10]

Enclosed pipe, carrying fluid or slurry, situated above ground or, in special conditions, below ground when marked by a cleared right-of-way and signage.

C10 Pipeline, major category used alone

Power Transmission Line [C20]

High voltage electrical line, on towers, situated on cleared right-of-way.

C20 Power transmission line, major category used alone

Miscellaneous Ground Transportation with Special Characteristics [C30-C31]

A portion of a ground transportation system that has separately identifiable characteristics. This major category is for specialized transportation, usually confined to a local area, that is separate from other ground transportation.

C30 Other ground transportation that is not a pipeline or a power transmission line. The major category is used alone when the minor category could not be determined.

C31 Aerial tramway, monorail, or ski lift

Feature Class D, Landmark

Definition Applicable to Landmark. Landmark is the general name given to a cartographic or locational landmark, a land use area, and a key geographic location. A cartographic landmark is identified for use by an enumerator while working in the field. A land use area is identified in order to minimize enumeration efforts from where people are restricted or nonexistent. A key geographic location is identified in order to more accurately geocode and enumerate a place of work or place of residence. TIGER/Line(TM) files contain only cartographic landmarks or land use areas, if identified within the county area, but not key geographic locations.

Landmark With Category Unknown [D00]

Source materials do not allow determination of the landmark category. This code should not, under most circumstances, be used since the source materials usually provide enough information to determine the major category.

D00 Landmark, major and minor categories unknown

Military Installation [D10]

Base, yard, or depot used by any of the armed forces or the Coast Guard

D10 Military installation or reservation, major category used alone

Multihousehold or Transient Quarters [D20-D29]

D20 Multihousehold or transient quarters, major category used alone when the minor category could not be determined

D21 Apartment building or complex

D22 Rooming or boarding house

D23 Trailer court or mobile home park

D24 Marina

D25 Crew of vessel

D26 Housing facility for workers

D27 Hotel, motel, resort, spa, YMCA, or YWCA

D28 Campground

D29 Shelter or mission

Custodial Facility [D30-D37]

This major category is for an institution that maintains guards, nurses, caretakers, and so forth to preserve the welfare of those individuals resident in the facility.

D30 Custodial facility, major category used alone when the minor category could not be determined

D31 Hospital

D32 Halfway house

D33 Nursing home, retirement home, or home for the aged

D34 County home or poor farm

D35 Orphanage

D36 Jail or detention center

D37 Federal penitentiary, State prison, or prison farm

Educational or Religious Institution [D40-D44]

D40 Educational or religious institution, major category used alone when the minor category could not be determined

D41 Sorority or fraternity

D42 Convent or monastery

D43 Educational institution, including academy, school, college, and university

D44 Religious institution, including church, synagogue, seminary, temple, and mosque

Transportation Terminal [D50-D55]

The facility where transportation equipment is stored, the destination for travel on the transportation system, or the intermodal connection facility between transportation systems.

D50 Transportation terminal, major category used alone when the minor category could not be determined

D51 Airport or airfield

D52 Train station

D53 Bus terminal

D54 Marine terminal

D55 Seaplane anchorage

Employment Center [D60-D66]

This major category is for a location with high density employment.

D60 Employment center, major category used alone when the minor category could not be determined

D61 Shopping center or major retail center

D62 Industrial building or industrial park

D63 Office building or office park

D64 Amusement center

D65 Government center

D66 Other employment center

Tower [D70-D71]

D70 Tower, major category used alone when the minor category could not be determined

D71 Lookout tower

Open Space [D80-D85]

This major category contains areas of open space with no inhabitants or with inhabitants restricted to known sites within the area.

D80 Open space, major category used alone when the minor category could not be determined

D81 Golf course

D82 Cemetery

D83 National park or forest

D84 Other Federal land

D85 State or local park or forest

Special Purpose Landmark [D90-D91]

Use this category for landmarks not otherwise classified.

D90 Special purpose landmark, major category used alone when the minor category could not be determined

D91 Post office box ZIP Code(R)

Feature Class E, Physical Feature

Physical Feature With Category Unknown

Source materials do not allow determination of the physical feature category. This code should not, under most circumstances, be used since the source materials usually provide enough information to determine the major category.

E00 Physical feature, tangible but not transportation or hydrographic. The major and minor categories are unknown.

Fence

This major category describes a fence that separates property. For example, a fence around a military reservation or prison separates the reservation from civilian land, thus, a fence line is a property line marked by a fence.

E10 Fence line locating a visible and permanent fence between separately identified property

Topographic Feature

This category refers to topographical features that may be used as boundaries or as a reference for an area. The Census TIGER data base contains topographic features used to define the limits of statistical entities in locations where no other visible feature could be identified.

E20 Topographic feature, major category used when the minor category could not be determined

E21 Ridge line, the line of highest elevation of a linear mountain

E22 Mountain peak, the point of highest elevation of a mountain

Feature Class F, Nonvisible Features

Definition Applicable to Nonvisible Features. Nonvisible features are used to delimit tabulation entities, property areas, and legal and administrative entities. The Census Bureau separately identifies nonvisible boundaries only when they do not follow a visible feature such as a road, stream, or ridge line.

Nonvisible Boundary With Classification Unknown or Not Elsewhere Classified [F00]

F00 Nonvisible boundary, major and minor categories unknown

Nonvisible Legal or Administrative Boundary [F10-F15]

This major category refers to nonvisible boundaries of legal or administrative areas.

F10 Nonvisible jurisdictional boundary of a legal or administrative entity, major category used when the minor category could
not be determined

F11 Offset boundary of a legal or administrative entity

F12 Corridor boundary of a legal or administrative entity

F13 Interpolated boundary of a legal or administrative entity used for closure through hydrological areas

F14 Superseded legal or administrative boundary

F15 Superseded legal or administrative boundary, corrected through post census process

Nonvisible Features for Data Base Topology [F20-F25]

This category contains various types of nonvisible lines used to maintain the topology in the Census TIGER data base.

F20 Nonvisible feature for data base topology, major category used when the minor category could not be determined

F21 Automated feature extension to lengthen existing physical feature

F22 Irregular feature extension, determined manually, to lengthen existing physical feature

F23 Closure extension to complete data base topological closure between extremely close features (used to close small gaps
between complete chains and create polygons to improve block labeling on cartographic products)

F24 Nonvisible separation line used with offset and corridor boundaries

F25 Nonvisible centerline of area enclosed by corridor boundary

Point-to-Point Line [F30]

F30 Point-to-point line, follows a line of sight and should not cross any visible feature, for example, from the end of a road to a
mountain peak.

Property Line [F40]

F40 Property line, nonvisible boundary of either public or private lands, e.g., a park boundary

ZIP Code(R) Boundary [F50]

F50 ZIP Code(R) boundary, reserved for future use in delineating ZIP Code(R) Tabulation Areas

Map Edge [F60]

F60 Map edge, now removed, used during data base creation

Nonvisible Statistical Boundary [F70-F74]

F70 Statistical boundary, major category used when the minor category could not be determined

F71 1980 statistical boundary

F72 1990 statistical boundary, used to hold collection and tabulation census block boundaries not represented by existing
physical features

F73 1990 statistical boundary and extent of land use, it is not classifiable as a physical feature

F74 1990 statistical boundary, used to hold a tabulation census block boundary not represented by an existing physical
feature

Nonvisible Other Tabulation Boundary [F80-F82]

F80 Nonvisible other tabulation boundary, major category used when the minor category could not be determined

F81 School district tabulation boundary

F82 Special census tabulation boundary

Feature Class H, Hydrography

Basic Hydrography [H00-H02]

This category includes shorelines of all water regardless of the classification of the water itself.

H00 Water feature, classification unknown or not elsewhere classified

H01 Shoreline of perennial water feature

H02 Shoreline of intermittent water feature

Naturally Flowing Water features [H10-H13]

H10 Stream, major category used when the minor category could not be determined

H11 Perennial stream or river

H12 Intermittent stream, river, or wash

H13 Braided stream or river

Man-Made Channel to Transport Water [H20-H22]

These features are used for purposes such as transportation, irrigation, or navigation.

H20 Canal, ditch, or aqueduct, major category used when the minor category could not be determined

H21 Perennial canal, ditch, or aqueduct

H22 Intermittent canal, ditch, or aqueduct

Inland Body of Water [H30-H32]

H30 Lake or pond, major category used when the minor category could not be determined

H31 Perennial lake or pond

H32 Intermittent lake or pond

Man-Made Body of Water [H40-H42]

H40 Reservoir, major category used when the minor category could not be determined

H41 Perennial reservoir

H42 Intermittent reservoir

Seaward Body of Water [H50-H53]

H50 Bay, estuary, gulf, sound, sea, or ocean, major category used when the minor category could not be determined

H51 Bay, estuary, gulf, or sound

H53 Sea or ocean

Body of Water in a Man-Made Excavation [H60]

H60 Gravel pit or quarry filled with water

Nonvisible Definition Between Water Bodies H70-H74]

The Census Bureau digitizes nonvisible definition boundaries to separate named water areas, for instance, an artificial boundary is drawn to separate a named river from the connecting bay.

H70 Nonvisible water area definition boundary, used to separate named water areas and as the major category when the minor
category could not be determined

H71 USGS closure line, used as maritime shoreline

H72 Census water center line, computed to use as median positional boundary

H73 Census water boundary, international in waterways or at 12-mile limit, used as area measurement line

H74 Census water boundary, separates inland from coastal or Great Lakes, used as area measurement line

H75 Census water boundary, separates coastal from territorial at 3-mile limit, used as area measurement line

Special Water Feature [H80-H81]

Includes area covered by glaciers or snow fields.

H80 Special water feature, major category used when the minor category could not be determined

H81 Glacier

Feature Class X, Not Yet Classified [X00]

Classification Unknown or Not Elsewhere Classified:

X00 Feature not yet classified

APPENDIX H – Underlay File Format

UNDERLAY FILE FORMAT

Each underlay image (256-color .BMP file) must have an associated .SAT file that contains the reference coordinates for the image. The ASCII text file contains the northwest corner coordinates followed by the southeast corner coordinates. The .SAT file must be saved with the same name as the graphic image, but with a .SAT extension (e.g. hoover.bmp would require a hoover.sat). The .SAT file layout is DD.dddd format imported as a double: UpperLON, UpperLAT, LowerLON, LowerLAT.

APPENDIX I – Autoload.Cty File Format

FORMAT OF AUTOLOAD.CTY

An external file, AUTOLOAD.CTY, may now be used to locate user defined place names on the map (to add places that may not be in the GNIS database used in our data). This is a plain ASCII file, two lines per place, with the structure described below.

Line-1: CTY "PlaceName" "State" Population Elevation
Line-2: Longitude Latitude

The quotes around PlaceName and State are optional, however, they are required if the PlaceName definition contains multiple words (separated with spaces, commas, etc.). State is the proper two-letter state abbreviation.

This file needs to be in the same path as the application, and it gets loaded automatically when the application is started.

The data becomes visible at scales below 2 miles, and is included in the search used by the ExecDialog, FindPlace and FindClosestPlace methods of the OCX.

APPENDIX J – CMX File Format

CMX FILE FORMAT

This file format is used to export/import user designed overlays from/to Precision Mapping Streets Ver 4.0 and 5.0, and the MAPPRO40.OCX (v4.00.03) and MAPPRO50.OCX mapping control from Chicago Map Corp.

The format consists of a header and a series of keyword "elements" with the required information for each element, as described below. Please, note that text between the dashed lines below is only for information purposes, and not part of the CMX file format.

HEADER Information

Required text at the beginning of the file. It describes the type of data file. Files created by subsequent releases of the program would possibly use a slightly modified header.

```
CHICAGO MAP EXCHANGE FORMAT v40
REAL
DATA
```

LINE Element

Defined by two end-points x1,y1 and x2,y2 and a set of brush and pen attributes. The pen and brush attributes are described at the end of the format description, prior to the sample file listing.

```
Line X1,Y1 X2,Y2
Pen(Style,Thickness,Color)
Brush(Pattern,Color)
```

PLINE (Polyline) Element

Defined by a number of contiguous straight line segments (two end-points each) and a set of brush and pen attributes. Up to 100 line segments may be joined. If X1,Y1 and Xn,Yn are identical, the Pline forms a closed polygon that may have a fill pattern (based on the Brush attributes). The pen and brush attributes are described at the end of the format description, prior to the sample file listing.

```
Pline N
X1,Y1
X2,Y2
.....
Xn,Yn
Pen(Style,Thickness,Color)
Brush(Pattern,Color)
```

ELLIPSE Element

Defined by a center point (x1,y1), a point on the circumference (x2,y2), and a set of brush and pen attributes. The pen and brush attributes are described at the end of the format description, prior to the sample file listing.

```
Ellipse
X1,Y1
X2,Y2
Pen(Style,Thickness,Color)
Brush(Pattern,Color)
```

BEZIER Element

Defined by two end-point (x1,y1 and x2,y2), an inflection point (x3,y3) and a set of brush and pen attributes. The pen and brush attributes are described at the end of the format description, prior to the sample file listing.

```
Bezier



X1,Y1
X2,Y2
X3,Y3
Pen(Style,Thickness,Color)
Brush(Pattern,Color)
```

TEXT Element

Defined by an X,Y coordinate, the user specified string (or road name when using the label command), the text size (in degrees latitude multiplied by 1.0E+06 and divided by 64 for internal unit consistency, see note below), text rotation angle (degrees, clockwise from due East), and a set of brush and pen attributes. The pen and brush attributes are described at the end of the format description, prior to the sample file listing.

* Note: The user should remember that the size given here is for the bounding box, including descenders, etc. For example, if the desired text bounding box size (height) was 0.35 degrees, then the CMC entry for the Text element size value would be: 0.35 * (1000000/64) = 5469

Also, note the 0,0 entry folowing the X1,Y1 coordinates below. It's reserved, required, and is always 0,0.

```
Text
X1,Y1
0,0
Str("Enter Sample Text here", TextSize, TextAngle)
Pen(Style,Thickness,Color)
Brush(Pattern,Color)
```

MCIRCLE Element

Mercator circle, defined by a center point (x1,y1), a point on the circumference (x2,y2), and a set of brush and pen attributes. The pen and brush attributes are described at the end of the format description, prior to the sample file listing.

```
Mcircle
X1,Y1
X2,Y2
Pen(Style,Thickness,Color)
Brush(Pattern,Color)
```

MLINE Element

Merrcator Line, defined by two end-points (x1,y1 and x2,y2), and a set of brush and pen attributes. The pen and brush attributes are described at the end of the format description, prior to the sample file listing.

```
Mline
X1,Y1
X2,Y2
Pen(Style,Thickness,Color)
Brush(Pattern,Color)
```

META Element

Symbol elements in the Precision Mapping Symbol library. Defined by an point (x1,y1), X and Y size of bounding rectangle (in degrees latitude and longitude, respectively, multiplied by 1.0E+06 and divided by 64 for internal unit consistency, similar to the Text element entry), the internal name of the individual metafile (built into the Precision Mapping library see note further down), a rotation angle (degrees, clockwise from due
East), and a set of brush and pen attributes. The pen and brush attributes are described at the end of the format description, prior to the sample file listing.

```
Meta X1,Y1 ScaleX,ScaleY Name("FileName.WMF",RotationAngle)
Pen(Style,Thickness,Color) Brush(Pattern,Color)
```

GROUP Element

Simply specifies that the preceding "N" elements are grouped together in a composite element.

```
Group N
Pen(Style,Thickness,Color)
Brush(Pattern,Color)
```

** Note that the Pen and brush information for this element is redundant and has NO effect on the ndividual Group Elements.

Pen(Style,Thickness, Color)

Pen Style: An index number pointing to line styles array shown below.

0: Solid
1: Dash
3: Dot
4: Dash-Dot
5: Dash-Dot-Dot
6: Clear
7: Inside Frame

Pen thickness: 0-15 pixels
Pen Color: 0-15 system colors

Brush(Pattern, Color)

Brush Pattern: Index pointing to fill pattern array.

0: Clear
1: Solid
3: Horizontal
4: VErtical
5: Forward Diagonal
6: Backward Diagonal
7: Horizontal Cross
8: Diagonal Cross

Brush Color: 0-15 system colors

Internal Meta file name

The symbol library distributed with Precision Mapping (PMAP.LIB), contains a total of 92 symbol meta files. The names of the files are 001.WMF to 092.WFM in the order in which they appear in the toolbox when invoked in Precision Mapping streets. For example, 009.WMF would refer to the symbol of the worker with the shovel, 018.WMF would be the hi-rise building, etc.

Sample CMX File #1

```
CHICAGO MAP EXCHANGE FORMAT v40
REAL
DATA
Pline 5
-88.485504,41.917952
-88.485504,41.910848
-88.477824,41.910848
-88.477824,41.917952
-88.485504,41.917952
 Pen(0,0,0)
 Brush(4,0)


Ellipse
-88.473460,41.913998
-88.469620,41.917838
 Pen(0,0,9)
 Brush(7,12)
Pline 4
-88.464128,41.916672
-88.468608,41.912896
-88.463104,41.910912
-88.464128,41.916672
 Pen(0,0,0)
 Brush(0,0)
Bezier
-88.460350,41.910745
-88.458773,41.917842
-88.462322,41.918334
 Pen(0,0,0)
 Brush(0,0)
Pline 4
-88.455424,41.912896
-88.452864,41.918848
-88.452096,41.913216
-88.449280,41.917440
 Pen(0,0,0)
 Brush(0,0)
Text
-88.489428,41.905521
 0.0, 0.0
 Str("Sample Text Entry",25,0)
 Pen(0,0,0)
 Brush(0,0)
Text
-88.477173,41.902645
 0.0, 0.0
 Str("SampleTextObject",0,0)
 Pen(0,0,0)
 Brush(0,0)
Meta
-88.467349,41.920996
36.62551,25.55604
 Name("003.WMF",339)
 Pen(0,0,0)
 Brush(0,0)
Meta
-88.460646,41.921094
48.22439,46.78055
 Name("002.WMF",0)
 Pen(0,0,0)
```

```
 Brush(0,0)
Line
-88.482478,41.924345
-88.478421,41.921235
 Pen(0,0,9)
 Brush(5,12)
Line
-88.491596,41.914885
-88.484105,41.920996
 Pen(0,0,0)
 Brush(0,0)
Pline 5
-88.498368,41.924096
-88.498368,41.918016
-88.491776,41.918016
-88.491776,41.924096
-88.498368,41.924096
 Pen(0,0,9)
 Brush(7,12)
Pline 4
-88.495040,41.918080
-88.497984,41.912960
-88.492096,41.912960
-88.495040,41.918080
 Pen(15,0,11)
 Brush(5,15)
Group 3
 Pen(6,1,0)
 Brush(5,0)
```

Sample CMX File #2

```
CHICAGO MAP EXCHANGE FORMAT
REAL
DATA

Line
-132.926549,57.492981
-57.271921,13.989185
 Pen(0,5,0)
 Brush(0,0)
Line
-132.831146,13.989185
-57.176518,57.302175
 Pen(0,5,0)
 Brush(0,0)
Ellipse
-94.956130,35.645680
-89.518178,41.083632
 Pen(0,3,12)
 Brush(0,0)
Text
-95.433146,26.582389
 0.0, 0.0
 Str("Sample Text (Yellow)",23851,0)
 Pen(0,3,14)
 Brush(5,0)
Pline 5
-110.411426,39.080190
-110.411426,32.688185
-103.351600,32.688185
-103.351600,39.080190
-110.411426,39.080190
 Pen(0,3,14)
 Brush(5,0)
Meta
-80.264059,36.504308
29533.95563,31956.29900
 Name("009.WMF",0)
 Pen(0,3,14)
 Brush(5,0)
Line
-113.220141,53.805239
```

```
-105.702091,46.287189
 Pen(0,0,0)
 Brush(0,0)
Line
-105.702091,46.287189
-101.525397,55.371500
 Pen(0,0,0)
 Brush(0,0)
Line
-101.525397,55.371500
-96.408946,45.347433
 Pen(0,0,0)
 Brush(0,0)
Line
-96.408946,45.347433
-91.501330,54.327326
 Pen(0,0,0)
 Brush(0,0)
Line
-91.501330,54.327326
-87.429052,45.138598
 Pen(0,0,0)
 Brush(0,0)
Line
-87.429052,45.138598
-81.894932,53.805239
 Pen(0,0,0)
 Brush(0,0)
Line
-81.894932,53.805239
-77.509403,45.973937
 Pen(0,0,0)
 Brush(0,0)
Pline 5
-134.312449,41.170738
-134.312449,27.387646
-118.858679,27.387646
-118.858679,41.170738
-134.312449,41.170738
 Pen(0,0,0)
 Brush(0,0)
Line
-127.420903,43.676755
-127.420903,24.672795
 Pen(0,0,0)
 Brush(0,0)
Bezier
-119.171931,41.066321
-134.416866,41.066321
-134.312449,27.492064
 Pen(0,0,0)
 Brush(0,0)
Group 3
 Pen(8,1,13)
 Brush(4,10)
```

APPENDIX K - Street Editing Mode

STREET EDITTING MODE

A mode was created in the OCX for adding/deleting or editing user defined street segments. The mode is invoked by setting MapMode=2.

f) When the mode is invoked, the cursor changes to a plain cross-hair, that can be used to locate vertices of a new street segment.

g) As the cursor moves around, a circle of "snap" or "attach" influence can be seen tracking the cursor movement always being on an existing road segment. If the cursor is inside this influence circle, when the left mouse button is clicked, then the current vertice will attach (snap) to the existing road point in the circle.

h) The user may continue to press the left mouse button and define new vertices (belonging to the same street polyline) at will. When the desired number of vertices have been defined, the user may press the right mouse button to signify completion of the current street polyline definition.

i) It should be noted that when the Street editing mode is invoked, all user-defined road segments become cyan for better/quicker identification.

j) When the right mouse button is pressed, a dialog appears that permits the user to specify the name for the created segment, as well as to assign the desired road attribute. The options available in this dialog are:

(File) New Clears all currently defined segments from memory (make certain you have saved any road segments you want, prior to selecting this command).

(File) Open Load a user specified external roads file, (see further down for file structure). Note that this operation will erase ALL user-defined segments currently in memory before loading the specified roads file (also see File, Merge.)

(File) Save The user may save the currently defined street segments to a file (the extension .STR is automatically appended)

(File) Merge Load a user specified external roads file, (see further down for file structure). Note that this operation will does NOT erase-user defined segments currently in memory but merges them with the ones loaded from the specified file.

(File) Exit Close the street editing dialog (note that this does NOT cancel the street editing mode, which can only be done by setting the appropriate MapMode value.)

(Options) ZoomAll Zooms the view port to the extents of all User Defined streets currently in memory.

(Options) Attach Toggles the display and operation of the "attach" circle on or off.

(Options) Ortho When toggled on, only horizontal and vertical street orientations are permitted.

Name This is the name assigned to the current road segment by the user. It's used to label the road segment, search for it, etc. (Note: When searching for streets, these road segments are identified as "User Defined" in the listbox that appears in the search dialog.)

Road Type Five Road types are allowed. The descriptions of these types in the dialog are self-explanatory.

Add [Button] Adds a newly defined road segment to the list of road segments already in memory. Note that these segments are NOT saved unless the File, Save command (from this dialog) is executed.

Modify [Button] Replaces the attributes of the currently selected road segment with new ones specified in the dialog.

Delete [Button] Deletes the currently selected road segment from the list of segments in memory (but not from a file such segments have been saved in, unless the File, Save command is executed subsequent to the deletion). Note the segments are selected by holding down the Shift when clicking the left mouse button.

Cancel [Button] Close the street editing dialog (note that this does NOT cancel the street editing mode, which can only be done by setting the appropriate MapMode value.)

k) A defined road segment is selected for editing, or deletion, by placing the cursor on the segment and pressing the left mouse button while holding down the Shift key. The selected road segment will assume flashing highlight attribute to clearly show the used that it is being modified

l) The user-created road segments are NOT visible at scales above the Tiger Street level scale, i.e. about 2 miles. Also, the editing mode ahould not be activated if the current scale is not at the Tiger street level scale or lower.

m) The number of road segments that can be loaded at any given time is 50,000.

n) User defined road segments may be "searched for" using the standard searching techniques of MAPPRO, by specifying the assigned street name.

o) While in the street editing mode, road segment vertices may be moved by placing the mouse cursor on them, and holding down the control key and the left mouse button. This action engages the vertex which may them be dynamically moved to a new location. When the mouse button is released, the new location of the vertex become permanent.

User-Created Street file format (plain text)

```
Street File:Chicago Map Corporation
STR "RoadName/SecondaryFileName" Class N
x1 y1
x2 y2
...
...
xN yN
```

where: *First Line is an identifying header line*

STR - Keyword used internally

RoadName - Name specified for the road by the user. Used for display and search purposes. Note that a secondary name may also be specified using the slash character as a separator.

Class - 50: Interstate
59: Primary Highway
68: Major Road
77: Minor Road
93: Ramp

N - Number of points for this Road segment

x1, y1 - The longitude and latitude (x and y) coordinates for each of the segments defining this road (in decimal degree units). It should be noted that streets files created with earlier releases of MAPPRO40.OCX, i.e., without the header line and using internal coordinates, may still be read by the OCX transparently.

Sample File

```
Street File:Chicago Map Corporation
 STR "I-440" 50 2
 -92.163200 34.775040
 -92.163072 34.780928
 STR "S70/Scenic Route" 59 5
 -92.199424 34.766656
 -92.194752 34.762752
 -92.187072 34.762816
 -92.184384 34.766912
 -92.187392 34.770560
 STR "Rt61/Main Street" 68 5
 -92.200192 34.775744
 -92.205056 34.775744
 -92.206400 34.777280
 -92.207616 34.781440
 -92.204352 34.782528
 STR "My Street" 77 8
 -92.201856 34.773888
```

```
-92.202688 34.774848
-92.206080 34.774592
-92.208384 34.776768
-92.209216 34.778112
-92.210432 34.777280
-92.208640 34.775296
-92.210304 34.774912
STR "Exit-001" 93 4
-92.196736 34.764416
-92.196736 34.766144
-92.196160 34.767424
-92.194944 34.769280
```

Using the AUTOLOAD.STR file

The AUTOLOAD.STR file, is a file containing user-defined streets, and as the name indicates, is automatically loaded when an application using MAPPRO40.OCX is started, if found in the default directory. It's loading is transparent.

Once the application is started, if the user invokes the street editing mode, it will be apparent that the AUTOLOAD.STR is loaded, as the New Road counter (next the dialog caption) will display the streets loaded from AUTOLOAD.STR.

If new streets are added, then the above counter will advance, and if the user saves the file, both the streets loaded from AUTOLAOD.STR and the newly added ones will be saved in the user-specified file. So, if the user re-started the application 5 times adding new streets in each session and saving each session to a different file, each file would also contain whatever streets were loaded from AUTOLOAD.STR.

That would be fine, if that's what the user intended to do. However, if the intent is to have different files containing new streets for "specific" areas, in order to avoid having the duplicity of the data in AUTOLOAD.STR in each of the generated files, here is what the user should do:

(1) If practical, the AUTOLOAD.STR street can be temporarily renamed, while the creation of such files takes place, or

(2) Prior to starting the generation of each of these "regional" or otherwise "specific" files, the Street editing mode should be started and the File, New command should be executed (making sure, of course, that any needed data is stored as desired by the user).

(3) When all new streets have been designed/created and saved to indiviual .STR files, the final step might be to add the changes to the AUTOLOAD.STR file. To do that:

   a. Exit the application.

   b. Restore the AUTOLOAD.STR name (if you want to add the changes to the existing AUTOLOAD.STR data)

   c. Restart the application

   d. Use File, Import (from the Streets dialog), to load and combine each individual street file.

   e. Save the combined data with the AUTOLOAD.STR file name.

   f. Exit and restart the application.

APPENDIX L - Delphi Code Examples

Delphi Code Examples

This appendix contains a number of Delphi source code snippets for performing useful operations, while using the MapPro.OCX.

Printing to A printer Using DirectDraw

```
//------------------------------------------------------
// Print Displayed Map using the current default printer
//------------------------------------------------------
var l,t,w,h,r,b,PixPerInX,PixPerInY:integer;
    margin:real;
    dc:hdc;
    syscolors:integer;
begin
  Label2.caption:='printing';
  application.processmessages; // let it paint the control

   // start the print job
   printer.begindoc;
   // Lock the printer device canvas to avoid problems
   printer.canvas.lock;

   dc:=printer.canvas.handle;
   // Get the horizontal and vertical resolution for the device
   pixperinx:=getdevicecaps(dc,logpixelsx);
   pixperiny:=getdevicecaps(dc,logpixelsy);
    syscolors:=getdevicecaps(dc,NUMCOLORS);//new for info
    label1.caption:=inttostr(syscolors);

   // L,T : Left and top margins, set to x.xx inches
   Margin:=0.7;
   L:=round(Margin*pixperinx);
   T:=round(Margin*pixperiny);

  // R,B : Right and Bottom margins, set to 2 inches
   R:=round(Margin*pixperinx);
   B:=round(Margin*pixperiny);

   // w,h: Width and height of print
    w:=printer.pagewidth-L-R;
    h:=printer.pageheight-T-B;
   // Make sure dimensions are at least 300x300 pixels
   if (w>300) and (h>300) then
   begin
     // Print color map: clear, scale, Mono
     MapPro1.directdraw(dc,L,T,W,H,true,true,true);
     messagebeep(31);
     // Print border around map
      with printer.canvas do
      begin
        brush.style:=tbrushstyle(bsclear);
        pen.width:=3;
```

```
        pen.color:=clblack;
       end;
       rectangle(dc,L,T,L+W,T+H);
    end;
    printer.canvas.unlock;
    printer.enddoc;
   label2.caption:='Done';
   label2.update;
end;
```

Saving to a Meta File

```
implementation

{$R *.DFM}

const inn:integer=0;
      SetUserPoint:boolean=false;

procedure TForm1.Button1Click(Sender: TObject);
var metadc,MetaHn:integer;
    d,metarect:trect;
begin
  // You could use this to create a memory file, if you wanted, and then
  // manipulate it as needed
  //  MetaDc:=CreateEnhMetaFile(canvas.handle,nil,nil,nil);
  //--
  // Or, this one so that you can create an external MetaFile
  // Create    an enhanced meta file canvas and get the dc to it
  MetaDc:=CreateEnhMetaFile(canvas.handle,'Test.wmf',nil,nil);
  //--
  // Use DirectDraw to draw the image on the metafile canvas.
  // (Note that you need to set the appropriate options, depending on
  // aht your desired outcome is) This is without scaling.
  MapPro1.DirectDraw(MetaDc,0,0,MapPro1.width,MapPro1.height,true,false,false);
  // Close the metafile
  MetaHn:=CloseEnhMetaFile(MetaDc);
  // Set the dimension of the PaintBox so  you can play the metafile.
  setrect(metaRect,0,0,MapPro1.width,MapPro1.height);
  d:=metarect;
  PlayEnhMetaFile(paintbox1.canvas.handle,MetaHn,d);
  // Delete the enhanced-format metafile handle.
  DeleteEnhMetaFile(metaHn);
end;

procedure TForm1.Button2Click(Sender: TObject);
var metadc,MetaHn:integer;
    d,metarect:trect;
begin
  // You could use this to create a memory file, if you wanted, and then
  // manipulate it as needed
  //  MetaDc:=CreateEnhMetaFile(canvas.handle,nil,nil,nil);
  //--
  // Or, this one so that you can create an external MetaFile
  // Create    an enhanced meta file canvas and get the dc to it
  MetaDc:=CreateEnhMetaFile(canvas.handle,'Test2.wmf',nil,nil);
  //--
  // Use DirectDraw to draw the image on the metafile canvas.
```

```
  // (Note that you need to set the appropriate options, depending on
  // that your desired outcome is). This is WITH scaling to show that the
created
  // output is not grainny, i.e., it is vector, and thus scaled as a vector.

MapPro1.DirectDraw(MetaDc,0,0,MapPro1.width*2,MapPro1.height*2,true,true,false);
  // Close the metafile
  MetaHn:=CloseEnhMetaFile(MetaDc);
  // Set the dimension of the PaintBox so you can play the metafile.
  setrect(metaRect,0,0,MapPro1.width*2,MapPro1.height* 2);
  d:=metarect;
  PlayEnhMetaFile(paintbox1.canvas.handle,MetaHn,d);
  // Delete the enhanced-format metafile handle.
  DeleteEnhMetaFile(metaHn);
end;

procedure TForm1.MapPro1PaintAfter(Sender: TObject; dc: Integer);
var i:integer;
    x,y:real;
begin
  if SetUSerPoint=true then
  begin
    inc(inn);
    for i:=1 to inn do
    begin
      x:=-100-i*1;
      y:=40+i*1;
      MapPro1.DrawObject(dc,x,y,2,clblue);
    end;
    listbox1.Items.add(inttostr(i)+':  DC =  '+inttostr(dc)+'     x,y =
'+floattostr(x)+','+floattostr(y));
    SetUserPoint:=false;
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
 SetUserPoint:=true;
 MapPro1.redraw;
end;
```

Using DirectView

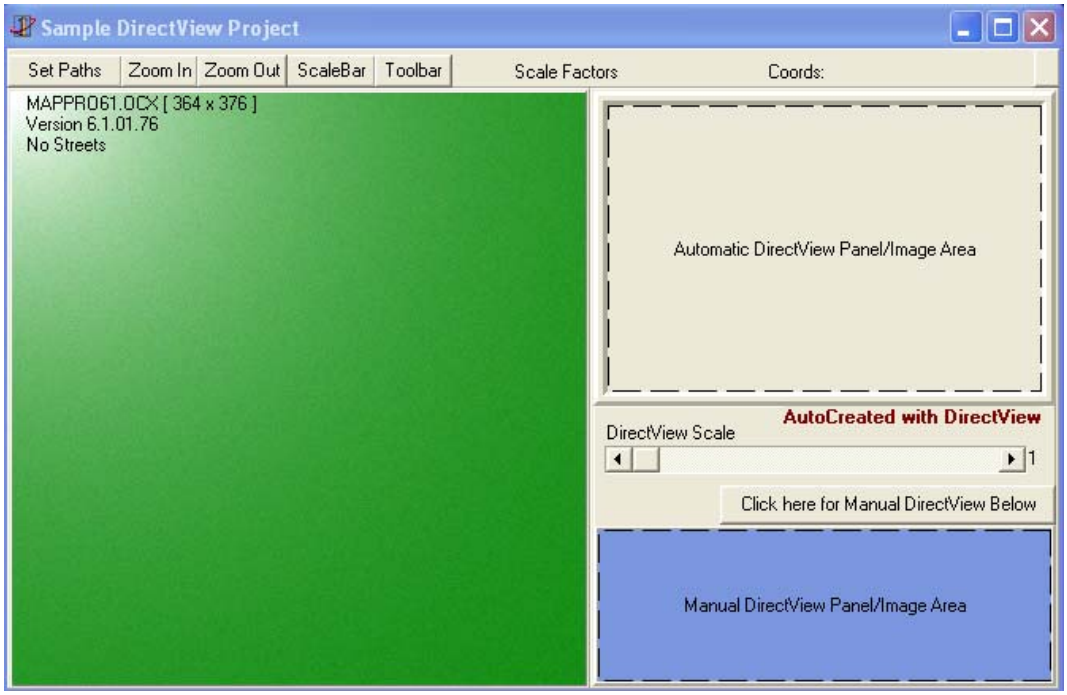
```
unit Unit_new1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, OleCtrls, MapPro61_TLB;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    Panel4: TPanel;
    MapPro1: TPmap6;
    Panel5: TPanel;
```

```
    Button1: TButton;
    Image1: TImage;
    ScrollBar1: TScrollBar;
    Label1: TLabel;
    Label2: TLabel;
    Image2: TImage;
    Label3: TLabel;
    Label4: TLabel;
    Button4: TButton;
    Button5: TButton;
    Button2: TButton;
    Button3: TButton;
    Label5: TLabel;
    Button6: TButton;
    Label6: TLabel;
    Panel6: TPanel;
    Image3: TImage;
    Button7: TButton;
    PaintBox1: TPaintBox;
    Image4: TImage;
    procedure ScrollBar1Change(Sender: TObject);
    procedure MapPro1PaintAfter(Sender: TObject; dc: Integer);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure MapPro1MouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
  private
    { Private declarations }
    Procedure WmDoWindow(var msg:tmsg);message wm_User+1;
  public
    { Public declarations }
    Procedure DoWindow1(image:timage;f:double);
    Procedure MakeSkin(scanvas:tcanvas;s:trect;dcanvas:tcanvas;d:trect);
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

Procedure Tform1.DoWindow1(image:timage;f:double);
var dc:integer; bmp:tbitmap; x1,y1,x2,y2:double; sf:double;
    sbefore_after:string;
begin
  if image.picture.bitmap.width=0 then
  begin
    bmp:=tbitmap.create;
    bmp.width:=image1.width;
    bmp.height:=image1.height;
    image.picture.bitmap:=bmp;
  end;

  image.picture.bitmap.canvas.lock;
```

```
  dc:=image.picture.bitmap.canvas.handle;
   with MapPro1 do
   begin
     x1:=lonleft;
     x2:=lonright;
     y1:=latbottom;
     y2:=lattop;
   end;
   sf:=MapPro1.miles /f;
   sbefore_after:=format('%6.2f',[x1]);
   MapPro1.DirectView(dc,image1.width,image1.height,x1,y1,x2,y2,sf);
   sbefore_after:=sbefore_after+' / '+format('%6.2f',[x1]);
   label6.caption:='In/Out: '+sbefore_after;
   image1.picture.bitmap.canvas.unlock;
   image1.Refresh;
end;


Procedure Tform1.WmDoWindow(var msg:tmsg);
begin
   DoWindow1(image1,scrollbar1.position);
end;

procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
  label1.caption:=Inttostr(scrollbar1.position);
  DoWindow1(image1,scrollbar1.position);
end;

Procedure Tform1.MakeSkin(scanvas:tcanvas;s:trect;dcanvas:tcanvas;d:trect);
var sx,sy,dx,dy:integer;
   ul,uls:trect;

   Procedure Transparent(scanvas:tcanvas;s:trect;dcanvas:tcanvas;d:trect);
   var r:trect; bmp:tbitmap;
   begin
     bmp:=tbitmap.create;
     bmp.pixelformat:=pf24bit;
     bmp.width:=(s.right-s.left);
     bmp.height:=(s.bottom-s.top);
     r:=s;
     offsetrect(r,-r.left,-r.top); // zero bse
     bmp.canvas.copyrect(r,scanvas,s);
     bmp.transparent:=true;
     bmp.transparentcolor:= clblack;
     bmp.transparentmode:=tmFixed ;

     dcanvas.draw(d.left ,d.top,bmp);
     bmp.free;
   end;

begin
  sx:=s.right-s.left;
  sy:=s.bottom-s.top;

  dx:=d.right-d.left;
  dy:=d.bottom-d.top;
  // upper left
  setrect(ul,0,0,dx div 2,dy div 2);
  uls:=ul;
```

```
  Transparent(scanvas,uls,dcanvas,ul);
 //- lower left
  setrect(ul,0,0,dx div 2,dy-dy div 2);
  uls:=ul;
  offsetrect(uls,0,sy-(ul.bottom-ul.top));
  offsetrect(ul,0,dy-(ul.bottom-ul.top));
  Transparent(scanvas,uls,dcanvas,ul);
  //-- upper right
  setrect(ul,0,0,dx-dx div 2,dy div 2);
  uls:=ul;
  offsetrect(uls,sx-(ul.right-ul.left),0);
  offsetrect(ul,dx-(ul.right-ul.left),0);
   Transparent(scanvas,uls,dcanvas,ul);
  //-- lower right
  setrect(ul,0,0,dx-dx div 2,dy-dy div 2);
  uls:=ul;
  offsetrect(uls,sx-(ul.right-ul.left),sy-(ul.bottom-ul.top));
  offsetrect(ul,dx-(ul.right-ul.left),dy-(ul.bottom-ul.top));
  Transparent(scanvas,uls,dcanvas,ul);
end;

procedure TForm1.MapPro1PaintAfter(Sender: TObject; dc: Integer);
var cv:tcanvas;
    d,s:trect;
    x1,x2,y1,y2,factor:double;
    dc3:integer;
begin
  MapPro1.SetItem(1,MapPro1.LonCenter, MapPro1.latcenter);
  MapPro1.SetItemBitmap(1,image4.picture.bitmap.handle);

  setrect(d,0,0,MapPro1.width,MapPro1.height);
  setrect(s,0,0,image2.picture.bitmap.width,image2.picture.bitmap.height);
  cv:=tcanvas.create;
  cv.handle:=dc;
  MakeSkin(image2.picture.bitmap.canvas,s,cv,d);
  cv.free;
  PostMessage(handle,wm_user+1,0,0);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MapPro1.ExecMethod('CONFIG');
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  MapPro1.ZoomIn;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
   MapPro1.ZoomOut;
end;

procedure TForm1.MapPro1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  label3.caption:=format('%8.6f,%8.6f',[MapPro1.xcord,MapPro1.ycord]);
end;
```

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  MapPro1.ScaleBar:=sbbottomleft;
end;
procedure TForm1.Button5Click(Sender: TObject);
begin
  MapPro1.toolbarmode:=tbfix;
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
  MapPro1.path_states0:='d:\mapdata\statesgdt';
  MapPro1.path_states1:='d:\mapdata\statesgdt';
  MapPro1.path_data1:='d:\mapdata\data1';
  MapPro1.path_data2:='d:\mapdata\data2';
  MapPro1.path_data3:='d:\mapdata\data3';
  MapPro1.DataSource:=2;

end;
procedure TForm1.Button7Click(Sender: TObject);
var x1,x2,y1,y2,factor:double;
    dc3:hdc;
begin
  dc3:=paintbox1.Canvas.Handle;
  paintbox1.Canvas.Lock;
     x1:=MapPro1.lonleft;
     x2:=MapPro1.lonright;
     y1:=MapPro1.latbottom;
     y2:=MapPro1.lattop;
     factor:=MapPro1.miles*0.4;

MapPro1.DirectView(dc3,PaintBox1.width,Paintbox1.height,x1,y1,x2,y2,factor);
     PaintBox1.canvas.unlock;
end;
end.
```



Delphi DirectView Project Form

APPENDIX M - Enumerated Properties & Record Structures

TxDegFormat
```
    dfDMS = 0
    dfDEC = 1
    dfDECMIN = 2
```

TxRasterOps
```
    RopBlack = 1
    RopNotMergePen = 2
    RopMaskNotPen = 3
    RopNotCopyPen = 4
    RopMaskPenNot = 5
    RopNot = 6
    RopXor = 7
    RopNotMaskPen = 8
    RopMask = 9
    RopNotXorPen = 10
    RopNop = 11
    RopMergeNotPen = 12
    RopCopy = 13
    RopMergePenNot = 14
    RopMerge = 15
    RopWhite = 16
```

TxMagnitude
```
    M500_0 = 0
    M300_0 = 1
    M200_0 = 2
    M100_0 = 3
    M050_0 = 4
    M030_0 = 5
    M020_0 = 6
    M010_0 = 7
    M005_0 = 8
    M003_0 = 9
    M002_0 = 10
    M001_0 = 11
    M000_5 = 12
    M000_3 = 13
    M000_2 = 14
    M000_1 = 15
```

TxMapMode
```
    MdZoom = 0
    MdUser = 1
    MdStreet = 2
    MdCad = 3
    MdDistance = 4
```

TxMapunit
```
    MuMi = 0
    Mukm = 1
```

TxTbMode
```
    TbHide = 0
    TbFix = 1
    TbFloat = 2
```

TxMouseButton
    mbLeft = 0
    mbRight = 1
    mbMiddle = 2

TxLLMode
    TmHMS = 0
    TmDEG = 1
    TmBIG = 2
    TmSMALL = 3

TxCoord
    CdLonlat = 0
    CdLatLon = 1

TxRteOps
    RT_Clear = 0
    RT_Spots = 1
    RT_Hatch = 2
    RT_Zoom = 4
    RT_Short = 16
    RT_Fast = 32
    RT_Direct = 48
    RT_Hours = 64
    RT_Km = 128
    RT_Total = 512
    RT_PrtMap = 1024
    RT_PrtDir = 2048
    RT_NoERR = 4096
    RT_NoDLG = 8192
    RT_Degree = 16384

TxAlignment
    taLeftJustify = 0
    taRightJustify = 1
    taCenter = 2

TxDragMode
    dmManual = 0
    dmAutomatic = 1

TxSourceData
    Z_NONE = 0
    ZP5_TIGER = 1
    ZPG_GDTUSA = 2
    ZPG_RESV1 = 3
    ZPG_GDTCDN = 4
    ZPG_RESV2 = 5
    ZPG_GDTUSACDN = 6

TxScaleBar
    sbNone = 0
    sbTopLeft = 1
    sbTopRight = 2
    sbBottomLeft = 3
    sbBottomRight = 4

TxRouteType
    Shortest = 0
    Fastest = 1
    Direct = 2
    Preferred = 3
    ShortUnbiased = 4

TxRasterOps
    RopBlack = 1
    RopNotMergePen = 2
    RopMaskNotPen = 3
    RopNotCopyPen = 4
    RopMaskPenNot = 5
    RopNot = 6
    RopXor = 7
    RopNotMaskPen = 8
    RopMask = 9
    RopNotXorPen = 10
    RopNop = 11
    RopMergeNotPen = 12
    RopCopy = 13
    RopMergePenNot = 14
    RopMerge = 15
    RopWhite = 16

TxDegFormat
    dfDMS = 0
    dfDEC = 1
    dfDECMIN = 2

Record TPointRec
    double x
    double y
    BSTR Name

Record TExtentRec
    double Xmin
    double Ymin
    double Xmax
    double Ymax

Record TrPoint
    double x
    double y

APPENDIX N - SAMPLE LICENSE AGREEMENT

MapOCX Pro 7.1, Rel. 2 Licensing & Distribution Agreement

This Agreement, effective _____, 2006, by and between UnderTow Software Corp., a Massachusetts corporation, having its principal place of business at 231 Sutton Street, Suite 2D-3, North Andover, Massachusetts, 01845 (hereinafter referred to as "USC") and _____ _____ having its principal place of business at _____ (hereinafter referred to as "LICENSEE").

WHEREAS, USC is the owner of certain mapping Software and Documentation, retains a license to certain Data from third party suppliers, and the copyrights for the work(s) listed in Exhibit A attached (hereinafter collectively referred to as the "USC Software" or "USC Copyrights" or "Copyrighted Works"); and

WHEREAS, LICENSEE desires to incorporate and/or employ USC Software and USC Copyrighted Works into its products pursuant to the terms of this Agreement; and

WHEREAS, LICENSEE desires to obtain and USC is willing to grant, a non-transferable, non-exclusive license to use the USC Software and related documentation in accordance with the terms and conditions of this Agreement; and

WHEREAS, LICENSEE is willing to obtain and USC is willing to grant the rights and licenses as provided in this Agreement.

NOW, THEREFORE, in consideration of the promises and mutual agreements herein contained, it is hereby agreed as follows:

1. DEFINITIONS

1.1. "COPYRIGHTS" means the common law Copyrights and the U.S. Copyright Registrations for the works, as described in the attached Exhibit A

1.2. "LICENSEE Products" or “Products” means custom applications developed by LICENSEE and any other product utilizing the Software, Data, Copyrighted Works or any portions thereof during the term of this Agreement.

1.3. "Licensed Software" or "Software" shall mean USC computer programs, data and data files in machine readable, object code form for the works listed in Exhibit A and any subsequent updates supplied by USC to LICENSEE pursuant to this Agreement, which may be amended from-time-to-time by the parties in writing.

1.4. “Directions” shall mean visual, verbal, and/or textual representation of a Route, Itinerary and/or Location(s).

1.5. “Location” shall mean a point identified by a longitude/latitude coordinate used to represent a physical location or the placement of a Mobile Unit etc. within the Enhanced Product(s). Location may be described in visual, verbal or textual terms.

1.6. "Map Display" shall mean a visual rendering of any portion of the Product(s) which may display lines, points or polygons that are sourced, in whole or in part, from the Product(s).

1.7. “Mobile Units” shall mean any mobile assets such as cars, vans, trucks, boats, trailers, buses, personnel, etc., equipped with the mobile component of the AVL Application. The mobile units need not be attached to the mobile asset, or be self-propelled or motorized.

1.8. “Route” shall mean the use of the Enhanced Product(s) to determine a logical means of progression from one Location to another.

1.9. “Transaction” shall mean a single distinct use of the USC Product to determine Location, Map Display, Route, or convey Directions to a user.

1.10. “Premier License” shall mean a license to use the Product(s), as incorporated into the Enhanced Products (a) in AVL Applications, Routing, scheduling and dispatch applications or route optimization applications any of which are used for the management of fleets of twenty five (25) or more Mobile Units, or (b) the developer receives periodic service fees from End Users, or (c) the application is delivered over the Internet or is server based.

1.11. “TANA’ or “TANA Data” shall mean third party data licensed by USC from Tele Atlas North America, Inc., as detailed in Exhibit A to this Agreement.

1.12. “Enhanced Product” shall mean the Licensed Software together with any product(s) and/or service(s) provided by the LICENSEE, and using the Licensed Software, or parts thereof.

1.13. “End User(s)” shall mean purchaser(s) of licenses to use the Enhanced Product for their own use and not with a view to the resale thereof. End User is further defined as a single, non-concurrent user of the Enhanced Product. The Enhanced Product is to be installed on a specific workstation, or licensed to a named person.

1.14. “Product(s)” shall mean USC's proprietary software and/or proprietary format databases(s) more particularly described in Exhibit A, attached to this Agreement.

2. GRANT OF LICENSE

2.1. For the consideration recited herein, USC grants to LICENSEE a non-exclusive, non-transferable right and license to copy and use the USC Licensed Software in products and to distribute and sell those Products to End Users.

2.2. No right or license is granted hereby by implication or otherwise to LICENSEE or any Third Party Manufacturer under any copyright, except as specifically provided herein.

2.3. No right or license is granted to copy, distribute or sell the Licensed Software unless packaged and bundled for sale by LICENSEE with USC's map engine and database.

2.4. The right and license granted is restricted and limited by the terms and conditions set forth in this Agreement, and in any other restriction set forth in Exhibits to this Agreement.

2.5. The Licensed Software may not be used for in-vehicle navigation applications meaning a permanently installed devices that provides real-time guidance and real-time map display to indicate the position of a vehicle, or other real-time information used to support turn-by-turn navigation.

2.6. In application involving tracking mobile assets, a single client seat license cannot be used to track more than twenty-five (25) such assets

2.7. Any distribution of the Enhanced Product shall be under a licensing Agreement between LICEBSEE and LICENSEE’s End User, that:

2.7.1. Includes a provision prohibiting the End User from translating and/or converting the data into another data format.

2.7.2. Includes a provision stating that if LICENSEE becomes insolvent, or otherwise ceases to exist, any End User payments and obligations will be directed to USC.

2.7.3. Includes a provision stating that if this License terminates, the End User License shall automatically terminate 12 months thereafter.

2.7.4. Includes the following: U.S. GOVERNMENT RIGHTS. If End User is an agency, department, or other entity of the United States Government, or funded in whole or in part by the United States Government, then use, duplication, reproduction, release, modification, disclosure or transfer of this commercial product and accompanying documentation, is restricted in accordance with the LIMITED or RESTRICTED rights as described in DFARS 252.227-7014(a)(1) (JUN 1995) (DOD commercial computer software definition), DFARS 227.7202-1 (DOD policy on commercial computer software), FAR 52.227-19 (JUN 1987) (commercial computer software clause for civilian agencies), DFARS 252.227-7015 (NOV 1995) (DOD technical data – commercial items clause); FAR 52.227-14 Alternates I, II, and III (JUN 1987) (civilian agency technical data and noncommercial computer software clause); and/or FAR 12.211 and FAR 12.212 (commercial item acquisitions), as applicable. In case of conflict between any of the FAR and DFARS provisions listed herein and this License, the construction that provides greater limitations on the Government's rights shall control. Contractor/manufacturer is Tele Atlas North America, Inc., 11 Lafayette Street, Lebanon, NH 03766-1445. Phone: 603.643. 0330. The Licensed Products are ©1984-2006 by Tele Atlas North America, Inc. ALL RIGHTS RESERVED. For purpose of any public disclosure provision under any federal, state or local law, it is agreed that the Licensed Products are a trade secret and a proprietary commercial product and not subject to disclosure.

2.7.5. Includes the following: If LICENSEE Partner is an agency, department, or other entity of any State government, the United States Government or any other public entity or funded in whole or in part by the United States Government, then LICENSEE Partner hereby agrees to protect the Licensed Products from public disclosure and to consider the Licensed Products exempt from any statute, law, regulation, or code, including any Sunshine Act, Public Records Act, Freedom of Information Act, or equivalent, which permits public access and/or reproduction or use of the Licensed Products. In the event that such exemption is challenged under any such laws, this LICENSEE Partner agreement shall be considered breached and any and all right to retain any copies or to use of the Licensed Products shall be terminated and considered immediately null and void. Any copies of the Licensed Products held by LICENSEE Partner shall immediately be destroyed. If any court of competent jurisdiction considers this clause void and unenforceable, in whole or in part, for any reason, this LICENSEE Partner agreement shall be considered terminated and null and void, in its entirety, and any and all copies of the Licensed Products shall immediately be destroyed.

3. DELIVERY

3.1. USC shall deliver to LICENSEE a master copy of the current version of the Licensed Software in a proprietary format object code form, suitable for reproduction, in electronic files only upon execution of this Agreement.

3.2. USC shall also deliver to LICENSEE one copy of the applicable printed Documentation for the Software.

3.3. LICENSEE acknowledges that Licensed Software requires user registration with USC for activation and LICENSEE may be required to establish a pre-paid royalty/licensing account

with USC for the purposes of activating licenses for products sold by LICENSEE, as provided in Exhibit "D" of this Agreement.

4. CONSIDERATION, LICENSE FEES AND PAYMENT

4.1. As consideration for the grant of this License, LICENSEE agrees to pay royalties to USC on all LICENSEE Products distributed or sold in accordance with the terms and conditions set forth in the attached Exhibit B.

4.2. If the tpe of Vendor Account selected by LICENSEE in Exhibit D does not require an escrow account, then on or before the 15TH day of each month of each year during the term hereof and/or the thirtieth (30th) day after termination hereof, shall send USC a written report in the format provided in the attached Exhibit C to USC. The report shall be certified by a financial officer of LICENSEE as to its correctness, and shall compute the amount due USC, for the preceding calendar month. Unless LICENSEE is otherwise directed in writing by USC, each such report shall be accompanied by a check in the proper amount then payable to USC as shown on such report, and sent to:

Undertow Software Corp.
Attention: Licensing
P.O. BOX 249
N. Andover, MA 01845

4.3. All amounts due and payable hereunder by LICENSEE shall be made in United States funds without deductions for taxes, assessments, fees, hold-backs or charges of any kind. Checks shall be made payable to USC and shall be forwarded to USC at the address in paragraph 4.2 above along with a Royalty Report in the form designated by USC.

4.4. LICENSEE shall keep complete and accurate records of all Products produced, distributed, sold or otherwise used or disposed of by LICENSEE with credit shown for returns for which actual credit is given to customers, such records to show separately the identity of and quantity of LICENSEE Product sold or otherwise used or disposed. Sales shall be considered as made on either the date of shipment of the products, or registration activation, if such registration is required under this License.

4.5. USC shall have the right, upon prior notice to LICENSEE, to have the correctness of any Royalty Report audited, at its expense, by an independent certified public accountant chosen by USC who shall examine LICENSEE's business records only on material pertinent to this Agreement. In the event the royalties reported and/or paid by LICENSEE are underreported by more than 5% as determined by the audit report, then LICENSEE shall immediately pay the balance owed plus audit expenses and an additional 20% of the amount owed as a penalty for said underreporting. The results of any audit shall remain confidential to the parties. USC shall have the right to terminate this Agreement if, after audit, LICENSEE has underreported royalties for any two months in a 12-month period. Royalty-based records of LICENSEE shall be kept available by LICENSEE for at least a period of time as such records are maintained in the ordinary course of business by LICENSEE after termination of the calendar year in which they are made.

4.6. LICENSEE shall pay USC interest at the rate of sixteen percent (16%) per annum, compounded monthly on any payment that is more than thirty (30) days overdue from the date such payment is due to the date of payment. This interest payment shall be in addition to any other remedy provided USC by law or this Agreement.

5. PROTECTION OF SOFTWARE

5.1. Proprietary Notices. LICENSEE agrees to respect and not to remove, obliterate, or cancel from view any copyright, trademark, confidentiality or other proprietary notice, mark, or legend appearing on any of the Software or output generated by the Software, and to reproduce and include same on each copy of the Software. Specifically, LICENSEE agrees to:

5.1.1. Have a prominent way for the End-User to access the "About" API provided with the Software, from the LICENSEE's End User interface.

5.1.2. Prominently display the information and any marks required and presented in Exhibit A to this Agreement.

5.2. No Circumvention or Reverse Engineering. LICENSEE agrees not to modify, reverse engineer, circumvent any technological measure used to control access to the copyright work, disassemble, or decompile the Software, or any portion thereof.

5.3. Ownership. LICENSEE further acknowledges that all copies of the Licensed Software and Copyrighted Works in any form provided by USC are the sole property of USC and/or its suppliers. LICENSEE shall not have any right, title, or interest to any such Licensed Software or Copyrighted Works or copies thereof except as provided in this Agreement, and further shall secure and protect all Licensed Software and Documentation consistent with maintenance of USC's proprietary rights therein.

5.4. Unauthorized duplication and/or Distribution. LICENSEE agrees not to be involved in any duplication and/or distribution of the Licensed Software or Copyrighted Works, or any part thereof, that is not expressly provided for in this Agreement.

6. CONFIDENTIALITY

6.1. Acknowledgment. LICENSEE hereby acknowledges and agrees that the USC Software and Documentation constitute and contain valuable proprietary products and trade secrets of USC and/or its suppliers, embodying substantial creative efforts and confidential information, ideas, and expressions. Accordingly, LICENSEE agrees to treat (and take precautions to ensure that its employees treat) the USC Software and Documentation as confidential in accordance with the confidentiality requirements and conditions set forth below.

6.2. Maintenance of Confidential Information. Each party agrees to keep confidential all confidential information disclosed to it by the other party in accordance herewith, and to protect the confidentiality thereof in the same manner it protects the confidentiality of similar information and data of its own (at all times exercising at least a reasonable degree of care in the protection of confidential information); provided, however, that neither party shall have any such obligation with respect to use or disclosure to others not parties to this Agreement of such confidential information as can be established to: (1) have been known publicly; (2) have been known generally in the industry before communication by the disclosing party to the recipient; (3) have become known publicly, without fault on the part of the recipient, subsequent to disclosure by the disclosing party; (4) have been known otherwise by the recipient before communication by the disclosing party; or (5) have been received by the recipient without any obligation of confidentiality from a source (other than the disclosing party) lawfully having possession of such information.

6.3. Injunctive Relief. LICENSEE acknowledges that the unauthorized use, transfer or disclosure of the USC Software and Documentation or copies thereof will: (1) substantially diminish the value to USC of the trade secrets and other proprietary interests that are the subject of this Agreement; (2) render USC's remedy at law for such unauthorized use, disclosure or transfer inadequate; and (3) cause irreparable injury in a short period of time. If LICENSEE breaches any of its obligations with respect to the use or confidentiality of the Software or Documentation, USC shall be entitled to equitable relief to protect its interests therein, including, but not limited to, preliminary and permanent injunctive relief.

6.4. Survival. LICENSEE's obligations under this Section will survive the termination of this Agreement or of any license granted under this Agreement for whatever reason.

7. INFRINGEMENT

7.1. LICENSEE shall notify USC of any infringement or threatened infringement by a third party of the USC Copyrights, or any of the Copyrights of USC's suppliers, which shall become known to it. USC shall be entitled to the exclusive right to use its own discretion in deciding how to act on this information, and its refusal to act thereon shall not affect this Agreement.

7.2. USC shall not be liable for any consequences or damage arising out of or resulting from anything made available hereunder or the exercise by LICENSEE of any rights granted hereunder nor be liable to LICENSEE for consequential damages under any circumstances.

8. WARRANTY

8.1. Ownership. USC represents and warrants that it owns free and clear of any liens or claims of ownership all of the proprietary rights licensed to LICENSEE.

8.2. Indemnity. LICENSEE agrees to indemnify and hold USC, its officers, directors, shareholders, representatives, agents and suppliers harmless from against any and all liabilities, damages, injuries, claims, suits, including USC's attorneys' fees incurred in its own defense (a) brought by any governmental authority concerning the Products or any portion thereof, or (b) that may in any way arise from breach of warranty, express or implied, or any tort actions or claims, as to the quality or defectiveness of the Products, or any portion thereof, or its merchantability, or its fitness for the purpose for which it was sold.

8.3. Notice of Proceedings. LICENSEE shall, within ten (10) days after receipt of notice of any legal proceeding against it with regard to the Products manufactured by or for LICENSEE, notify USC. LICENSEE shall have the right to defend any claim relating to production, distribution or consumption of the Products at its sole expense.

8.4. Limited Warranty. USC represents and warrants to LICENSEE that the Licensed Software, when properly installed by LICENSEE and used with LICENSEE's application, will perform substantially as described in USC's then current Documentation for such Software for a period of ninety (90) days from the date of shipment.

8.5. Limitations. Notwithstanding the warranty provisions set forth herein, all of USC's obligations with respect to such warranties shall be contingent on LICENSEE's use of the Software in accordance with this Agreement and in accordance with USC's instructions as provided by USC in the Documentation, as such instructions may be amended, supplemented, or modified by USC from time to time. USC shall have no warranty obligations with respect to any failures of the Software which are the result of accident, abuse, misapplication, extreme power surge or extreme electromagnetic field.

8.6. LICENSEE's Sole Remedy. USC's entire liability and LICENSEE's exclusive remedy shall be, at USC's option, either: (1) return of the price paid; or (2) repair or replacement of the Software upon its return to USC; provided USC receives written notice from LICENSEE during the warranty period of a breach of warranty. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

8.7. Disclaimer of Warranties. USC DOES NOT REPRESENT OR WARRANT THAT ALL ERRORS IN THE SOFTWARE AND DOCUMENTATION WILL BE CORRECTED. THE WARRANTIES STATED IN THIS SECTION ARE THE SOLE AND THE EXCLUSIVE WARRANTIES OFFERED BY USC. THERE ARE NO OTHER WARRANTIES RESPECTING THE SOFTWARE AND DOCUMENTATION OR SERVICES PROVIDED HEREUNDER, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF DESIGN, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE, EVEN IF USC HAS BEEN INFORMED OF SUCH PURPOSE. NO AGENT OF USC IS AUTHORIZED TO ALTER OR EXCEED THE WARRANTY OBLIGATIONS OF USC AS SET FORTH HEREIN.

8.8. Limitation of Liability. LICENSEE ACKNOWLEDGES AND AGREES THAT THE CONSIDERATION WHICH USC IS CHARGING HEREUNDER DOES NOT INCLUDE ANY CONSIDERATION FOR ASSUMPTION BY USC OF THE RISK OF LICENSEE'S CONSEQUENTIAL OR INCIDENTAL DAMAGES WHICH MAY ARISE IN CONNECTION WITH LICENSEE'S USE OF THE SOFTWARE AND DOCUMENTATION. ACCORDINGLY, LICENSEE AGREES THAT USC SHALL NOT BE RESPONSIBLE TO LICENSEE FOR ANY LOSS-OF-PROFIT, INDIRECT, INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE LICENSING OR USE OF THE SOFTWARE OR DOCUMENTATION.

Any provision herein to the contrary notwithstanding, the maximum liability of USC to any person, firm or corporation whatsoever arising out of or in the connection with any license, use or other employment of any Software delivered to LICENSEE hereunder, whether such liability arises from any claim based on breach or repudiation of contract, warranty, tort or otherwise, shall in no case exceed the actual price paid to USC by LICENSEE for the Software whose license, use, or other employment gives rise to the liability. The essential purpose of this provision is to limit the potential liability of USC arising out of this Agreement. The parties acknowledge that the limitations set forth in this Section are integral to the amount of consideration levied in connection with the license of the Software and Documentation and any services rendered hereunder and that, were USC to assume any further liability other than as set forth herein, such consideration would of necessity be set substantially higher.

9. COPYRIGHT OWNERSHIP AND USE

9.1. LICENSEE shall not use the USC Copyrights in connection with any product other than the Licensed Products.

9.2. LICENSEE agrees to apply appropriate legends on its labels or other printed materials indicating ownership by USC, and/or its suppliers, and, as appropriate, registration of said Copyrights, as provided in Exhibit A of this Agreement.

9.3. Neither this Agreement, the facts of this Agreement or its termination, or LICENSEE's exercise of the rights granted under this Agreement, including use of said Copyrights, shall effect, diminish or otherwise alter USC's right, title and interest in and to the Copyrights and LICENSEE agrees to assign, and hereby does assign to USC any and all right, title and interest in said Copyrights which may accrue to or otherwise be acquired by LICENSEE

during or as a result of this Agreement or any other undertaking or contract between the parties hereto. Nothing in this Agreement shall be construed to grant or otherwise convey to LICENSEE any ownership or any like right to LICENSEE relating to said Copyrights.

9.4. LICENSEE acknowledges USC's ownership of and exclusive rights in the Copyrights and agrees that it will not use the Copyrights except as authorized hereunder or as may hereafter be agreed by USC in writing. LICENSEE agrees that the benefits of its use of said Copyrights shall inure solely to the benefit of USC. LICENSEE agrees to execute and deliver to USC such documentation relating to the Copyrights as may be required by law, or be requested by USC, including, but not limited to, any documents needed in conjunction with any trademark registration, application for registration, renewal or registered user agreement.

9.5. LICENSEE acknowledges that all Copyrights to TANA Data is owned by Tele Atlas North America, Inc., and its suppliers, as described in Exhibit A to this Agreement, and agrees to display the appropriate Copyright notices provided in Exhibit A.

10. PRODUCT QUALITY

10.1. LICENSEE agrees that Products sold pursuant hereto shall conform to standards of quality established by any state or federal agency having jurisdiction over said products.

10.2. LICENSEE agrees, at least once every year and at USC's request, to supply USC with a copy of all of LICENSEE's current Products, using the Software, to permit USC to determine if quality standards and licensing requirements are being maintained LICENSEE.

10.3. LICENSEE agrees to supply USC, upon request, with copies of written material including packaging, advertising, labels and the like on, or in, which the Copyright Notices appear, to permit USC to determine if LICENSEE's use of the Copyrights are in accordance to this Agreement.

11. TERM AND TERMINATION

11.1. The initial term of this Agreement shall begin on the date of acceptance by LICENSEE and continue for one (1) year and shall automatically continue in full force and effect for an additional one (1) year period, unless sooner terminated as hereinafter provided, or written notice of termination is provided to the other party at least 90 days prior to expiration of the term.

11.2. USC may terminate this Agreement upon written notice to LICENSEE if LICENSEE remains in default in making any payment or report required hereunder, or fails to comply with any other provision hereof for a period of fifteen (15) days after written notice of such default or failure is given by USC to LICENSEE. USC shall also have the right to place LICENSEE'S account on hold, and prohibit registrations, if LICENSEE is overdue on any payment for more than 30 days.

11.3. Any termination of this Agreement shall not relieve LICENSEE of liability for any payments accrued prior to the effective date of such termination, or for any payments on Products marked with said Copyrights pursuant to this Agreement prior to the effective date of such termination and sold thereafter.

11.4. Upon termination of this Agreement, LICENSEE agrees to cease and desist from any and all further use of USC Copyrights, or use of any of USC suppliers Copyrights and data.

11.5. Upon termination of this Agreement, USC shall be entitled to all relief in law and equity, including but not limited to, entry of judgment on the amount of consideration then due and

owing, preliminary and permanent injunctive relief, as well as USC's costs and attorney's fees incurred in any enforcement action.

12. ASSIGNMENT

12.1. This Agreement may not be assigned by LICENSEE without written authorization by USC. Such authorization shall not be unnecessarily withheld.

12.2. This Agreement shall inure to the benefit of, and be binding upon, the successors and assigns of both parties, but no assignment by LICENSEE of this Agreement or any part thereof, including any transfer or the like in conjunction with a sale, merger or the like of LICENSEE or any part thereof, shall have any force or validity whatsoever, except, unless and until approved in writing by USC.

13. MISCELLANEOUS

13.1. All notices or communications which either party may desire, or be required, to give to the other shall be in writing and shall be deemed to have been duly served if and when forwarded by registered or certified mail or overnight delivery by Federal Express to such address as shall have been designated by notice from the addressee for addressing of notices to it, or if no such designation shall have been made, to the address of the party appearing below:

(LICENSOR):	(LICENSEE):
Undertow Software Corp. Attention: Licensing P.O. Box 249 N. Andover, MA 01845	_____ _____ _____ _____

13.2. The failure to act upon any default hereunder shall not be deemed to constitute a waiver of such default.

13.3. The validity, legality and enforceability of any provision hereof shall not be affected or be impaired in any way by any holding that any other provision contained herein is invalid, illegal or unenforceable in any respect.

13.4. LICENSEE hereby assumes responsibility for obtaining all necessary official governmental approval for the Products produced and sold under this Agreement.

13.5. Dispute Resolution. All disputes other than non-payment of consideration set forth in Section 4 which arise in connection with performance under this Agreement, at USC's discretion may be first submitted to good faith, face-to-face mediation with a qualified mediator chosen by the parties and located in Boston, Massachusetts before any party shall be entitled to file any legal action hereunder. This Agreement and any mediation, arbitration or litigation hereunder shall be governed by the rules and laws of the State of Massachusetts. Termination for non-payment of consideration under paragraph 4 shall entitle USC to seek immediate relief from the court in accordance with the terms of this Agreement.

13.6. Jurisdiction. If the parties are unable to resolve any dispute by good faith negotiations or mediation as set forth above, the parties agree that the exclusive venue for all actions relating in any manner to this Agreement shall only be brought in the courts of the Commonwealth of

Massachusetts, and each party consents and submits to the personal jurisdiction of such courts (and of the appropriate appellate courts) in any such action or proceeding and irrevocably waives any objection to venue laid therein. Process in any action or proceeding referred to in the preceding sentence may be served on any party anywhere in the world.

13.7. This Agreement and the terms thereof shall be construed, interpreted, applied and enforced under and pursuant to the laws of the State of Massachusetts.

14. INTEGRATION

This Agreement constitutes the entire understanding of the parties, and revokes and supersedes all prior agreements between the parties and is intended as a final expression of their Agreement. It shall not be modified or amended except in writing signed by the parties hereto and specifically referring to this Agreement. This Agreement shall take precedence over any other documents that may be in conflict therewith.

The LICENSEE warrants and represents that the signator or person accepting this License is an officer of LICENSEE and is authorized to enter into this Agreement on behalf of the LICENSEE.

IN WITNESS WHEREOF, the parties have executed this Agreement effective as of the date first written above.

UNDERTOW SOFTWARE CORP.	LICENSEE:
By (Name)_____	By: _____
Title: _____	Title: _____
Signature: _____	Signature: _____
Date:_____	Date: _____

LICENSING EXHIBIT A
LICENSED PRODUCT AND COPYRIGHTS DISPLAY

Licensed Software/Data:

- MapOCX Pro 7.1, Release 2 (MapPro71.OCX, Release 2)
- Tiger 2000 Data for the United States Dataset (Depending on License)
- TANA Data for the United States (Depending on License)
- TANA Data for Canada (Depending on License)

Copyrights

USC - USC is the owner of certain mapping Software, Data and Documentation and the common law copyrights for the work entitled "Map OCX Pro 7.0 or MapOCX Pro 7.1, and United States Copyright Certificates of Registration for the works entitled: "Map OCX Pro 4.0", Registration No. ATX 5-571-581, effective August 16, 2002; and "Precision Mapping Streets 3.0", Registration No. TX 4-526-524, effective April 21, 1997, (hereinafter referred to as the "USC Software" or "USC Copyrights" or "Copyrighted Works").

Required displays in End-User Products

LICENSEE shall conspicuously display each applicable copyright notice for the Licensed Software on the initial splash screen, in the code, on the storage medium, on the packaging, in the "Help/About" section, in the operator's manual, and in any displayed or printed map image.

On All Products

© 2006 UnderTow Software Corp.

In addition, on Products Using the TANA Data set for the USA and/or Canada

Data © 2006 Tele Atlas North America, Inc.

In addition, on Products Using the TANA Data set for Canada

© 2006 Geographic Data Technology (Canada) Inc. All rights reserved. This material is proprietary and the subject of copyright protection and other intellectual property rights owned or licensed to Geographic Data Technology (Canada) Inc. The use of this material is subject to the terms of a License Agreement. You will be held liable for any unauthorized copying or disclosure of this material."

In addition, on Products Using the Points of Interest (POI)

LICENSING EXHIBIT B
CONSIDERATION AND PAYMENT TERMS
RESTRICTIONS AND LIMITATIONS TO LICENSE

The following pricing applies to each individual dataset: Tiger 2000 covering the United States, TANA data covering the United States, and TANA data covering Canada. For each instance of the dataset distributed, used or registered, LICENSEE will compensate USC based upon the following guidelines. Unless otherwise noted, these are one-time fees and entitle the end-user to perpetual use of the version of the Software and Data provided to LICENSEE at the time of the execution of this Agreement.

(1) Standard Licensing – LICENSEE is Distributing End-user Application based on the Software

LICENSEE will pay USC for each user client seat distributed or registered. Each single user client seat license must be either paid for at the time of distribution or at the time of registration of the software. If the LICENSEE has escrow account funds available at the time of registration, the licensing fee will be automatically deducted from the account automatically; in the event that sufficient funds are not available, the registration system will either prompt the user to purchase the client seat license or prohibit the registration process entirely. It is the LICENSEE's sole responsibility to contact USC to establish the desired account, and to insure that funds are available to register any client seats.

All license purchases are non-refundable. Should LICENSEE dispute a registration or deduction from their escrow account, LICENSEE has 30 days from the date of the charge to refute these charges. After 30 days, all transactions are considered final.

Volume discount pricing is available based upon the quantity purchased for each order, at a given time; pricing is not cumulative based upon past purchases.

For Applications that DO NOT provide Routing and/or Directions capability the following fees shall apply:

Number of Client Licenses (Seats**) Purchased	License Cost per Client Seat **TANA USA** Dataset Only	License Cost per Client Seat **TANA USA** Dataset Only **w/POI**	License Cost per Client Seat **TANA USA & Canada** Datasets	License Cost per Client Seat **TANA USA & Canada** Datasets **w/POI**	License Cost per Client Seat **Tiger** 2000 Data set
1 - 24	$120.00	$150.00	$180.00	$210.00	$100.00
25 - 49 *	$100.00	$125.00	$150.00	$175.00	$80.00

*** Contact Undertow Software Corp. for Higher Volume Pricing (as low as $20 ea)**

For Applications that **DO provide** Routing and/or Directions capability the following fees shall apply:

Number of Client Licenses (Seats**) Purchased	License Cost per Client Seat **TANA USA** Dataset Only	License Cost per Client Seat **TANA USA** Dataset Only **w/POI**	License Cost per Client Seat **TANA USA & Canada** Datasets	License Cost per Client Seat **TANA USA & Canada** Datasets **w/POI**	License Cost per Client Seat **Tiger** 2000 Data set
1 - 24	$160.00	$190.00	$220.00	$250.00	$100.00
25 - 49 *	$133.00	$175.00	$183.00	$225.00	$80.00

*** Contact Undertow Software Corp. for Higher Volume Pricing (as low as $25 ea)**

*** Number of Seats shall mean the number of client seat licenses purchased at one time.*

(2) Premier Licensing –Per Mobile Asset being Tracked or Per Transaction Pricing

(a) Tracking Mobile Assets Licensing

i. If the application is used *in house*, using the 4 client seat licenses provided as part of the Software License, to track mobile assets, whether these assets belong to LICENSEE or LICENSEE's customers, and provided that LICENSEE does NOT charge any monthly, or other fees, or consideration, for such tracking. LICENSEE can track up to twenty-five (25) mobile assets per client seat, without ANY additional payment to Undertow. If additional client seats are needed to meet the 25 asset requirement, client seat licenses need to be purchased, at the prices described in section (1) of this Exhibit B.

ii. If the application is used *in house* as outlined in (i) above, and LICENSEE charges a fee or consideration, for tracking the mobile assets, then a licensing fee of "$2 - per month - per asset being tracked", provided that the application does not involve the routing or directions, or a fee of "$3.50 - per month - per asset being tracked", if the application involves routing or directions, shall also paid to Undertow.

iii. If LICENSEE distributes copies of their application to end-users/clients, then each such copy requires a client seat license, at the prices describer in section (1) of this Exhibit B, and each such licensed copy is subject to the same limitations and conditions, as described in (i) and (ii) above. *For example*, if LICENSEE develops an application and licenses it to a client to track their assets, then that client needs a client license (purchased either by LICENSEE or by customer at the time of registration). Client can then use such license to track their own assets (up to 25 per client seat), no additional payment is required. If LICENSEE's client uses licensed software to track other assets, for which they charge a fee or consideration, then LICENSEE shall pay a fee of "$2 - per month - per asset being tracked", provided that the application does not involve the routing or directions, or a fee of "$3.50 - per month - per asset being tracked", if the application involves routing or directions.

***Note** that the above pricing for Premiere Licenses does NOT include Map Display in Mobile Units.*

(b) Server or Internet Server Licensing

Server or Internet Server Licensing is provided to LICENSEE on a "Per Transaction" basis. The number of Transactions permitted for each LICENSEE on an annual basis, will be determined on the number of Transactions purchased by LICENSEE at the beginning of each year. At the end of each year, the total number of transactions will be tallied and LICENSEE shall pay for any transaction above and beyond those purchased at the beginning of the year.

# of Transactions Purchased at Beginning of Year *	**Annual Licensing Cost**
125,000	$2,000.00
500,000	$5,000.00

*** For Higher Transaction numbers and discounts, contact Undertow**

EXHIBIT C - ROYALTY REPORT
ROYALTY REPORT FOR UNDERTOW SOFTWARE CORP.

LICENSEE:

*Name:*_____

*Address:*_____

*Address:*_____

*City:*_____ *State:*_____

*Country:*_____

Remit To:

Undertow Software
PO BOX 249
N. Andover, MA 01845
Tel 978-794-9377 Fax 978-688-6312

Reports the following information for _____ 20___:
(Month) (YY)

Royalty Per Mobile Unit:

a. Total Units: _____

b. Fee Per Unit: _____

c. Total Royalty Fee: _____

(OR)

Royalty Per Unit

a. Total Units Sold: _____

b. Royalty Fee Per Unit: _____

c. Total Royalty Fee: _____

(OR)

Internet Transaction Count Number: _____

Amount Enclosed: _____ Check Number: _____

*If there were no royalties for this month, please leave a brief explanation:

_____

_____

_____

I _____ am acting as an authorized representative of the company listed above. The information is accurate and true, which meets the criteria of our agreed upon contract.

_____ _____/_____/_____
Signature Date

LICENSING EXHIBIT D
VENDOR ACCOUNTING AND TYPE OF DEPLOYMENT
(This form needs to be signed as part of the Agreement)

I. Type of Vendor Account Desired (please select one)

___ **Per Transaction Fee** - Special licensing for applications that are based on a single, or multiple servers that are used to dispense maps, directions, etc. to clients through a wide LAN or web/internet connection. Please, note that you need a special Licensing Agreement for this type of deployment. POIs cannot be part of this type of license.

___ **Pre-Pay Only** - Each vendor has an escrow account on the USC registration servers. Money can be deposited into the account by the vendor at any time, by logging to www.registermyapp.com/login.asp and making an on-line payment using their credit card, or by mailing USC a check for deposit into their account. When an end user client seat is about to be registered, the escrow account is checked and if the balance is sufficient to process the registration, it goes through, and a serial Registration code is issued. If the balance is insufficient, a message is displayed in the end-user's registration dialog to contact USC. Note that this is the preferred license, in particular if vendors want to pre-purchase a number of licenses so that they can get a volume discount. It may also be preferred because it allows the end-user to register themselves, without any intervention by the vendor.

___ **Pre-Pay Register by Vendor** - Same as option (d) above, but the registration can be done ONLY by the vendor (if vendors require such control). The vendor needs to log on to the USC registration servers (using their Vendor Code and Vendor Accounting Password) to process each registration.

___ **Trial Version (Sell it)** - This is probably the simplest implementation from the developer/vendor standpoint. The vendor develops and deploys their application and at the time of the client seat registration, either the end user or the vendor can pay for it, using our on-line purchase system (they will be prompted appropriately and be directed to UnderTow site's shopping cart)

II. Type of Application Developed and Distributed by LICENSEE (select all the apply)

___ **No Routing Directions –** The application deployed by LICENSEE to the end user, does not provide any routing directions, and is not used to track mobile assets.

___ **With Routing Directions –** The application deployed by LICENSEE to end users, includes the ability to provide routing directions, and is not used to track mobile assets.

___ **No End-User application, No Tracking –** The application is used in-house by LICENSEE, only. There is no distribution to end users, and it is not used to track mobile assets.

___ **No End-User application, With Tracking –** The application is used in-house by LICENSEE, only. There is no distribution to end users, and it is used to track mobile assets.

___ **End User Asset Tracking –** The application deployed by LICENSEE to the end user, is used to track mobile assets, by each end-user.

LICENSEE:

Signature:_____

Title: _____

APPENDIX O – DEPLOYING APPLICATIONS DEVELOPED WITH THE MapPro71.OCX, Release2 SDK

A number of developers were confused about the requirements/details for deploying their end-user/client applications based on MapPro71 Rel 2, and although portions of this information appear in various sections in this manual, it is also summarized here for convenience.

Note – Deployment cannot take place without a Signed Licensing and Distribution Agreement!

One of the most important considerations in deploying applications using MapPro71 Rel 2, is whether the application uses point-to-point routing or not. End-user license pricing is different for applications that use the routing capabilities of the OCX. If the deployed applications does not use routing, then you need to make sure that the property **RoutingActive** is set to False, otherwise the higher prices, including routing will be charged at the time of registration of the client license.

1. In deploying your end-user Product, you need to distribute (copy or install to the end user's system HD) the following

    a. A copy of your application (with an appropriate installer if applicable, to get the rest of the material referenced here, installed to your customer's system)

    b. A copy of the MapPro71.OCX mapping control

    c. A copy of the data folders: DATA1, DATA2, DATA3, STATES that were provided to you when you purchased the MapPro71 SDK (the appropriate files need to be copied depending on whether you are deploying Tiger or Premium TeleAtlas Dynamap® data).

    Important Note: End-User License Registration, Proper operation of each installed/registered copy and the cost of each registration, that is automatically determined by our registration servers, depend on the signature (*.SIG) files present in the STATES and in the DATA4 folder, if the Points of Interest (POI) database is also licensed for distribution.

        (1) If your application is going to be using the US Tiger data, and that is the only dataset you want to be charged for, at the time of the registration, then the only SIG file in that folder should be TGR01.SIG.

        (2) If your application is going to be using ONLY the Dynamap® Data for the USA (not Canada), and that is the only dataset you want to be charged for, at the time of the registration, then the only SIG file in that folder should be TANAUSA07.SIG.

        (3) If your application is going to be using ONLY the Dynamap® Data for CANADA (not the USA), and that is the only dataset you want to be charged for, at the time of the registration, then the only SIG file in that folder should be TANACAN08.SIG.

        (4) If your application is going to be using the Dynamap® Data for the USA and CANADA, then both TANA07.SIG and TANA08.SIG files need to be in the STATES folder.

(5) If your application is also going to be using the POI database, then the signature file POI0A.SIG should be in the DATA4 directory (which contains the POI data).

Please note that if other SIG files (for example, SIG files from earlier releases) are present in the states folder during the registration process, you vendor account will be charged for licenses to all of them. That's why it is important to distributed ONLY the SIG files needed and make sure you delete any other SIG afiles.

d. Any other files that your application may need.

Note: *You do not need to purchase and or distribute a copy of UnderTow's end-user product Precision Mapping Streets and Traveler, for each client license, to get access to the mapping data. The data is in the folders described in 1.c above, and you are licensed to distribute copies of this data set under one of the Licensing Account Agreements described below.*

2. Before you start deploying your end-user application you need to decide what type of a Licensing Account you want, and you **need** to sign the Licensing & Distribution Agreement (usually an appendix of the User's manual for each mapping control). Remember, as explained in the MapPro user's manual, each client (end-user) installation needs to have a client license whish is obtained though registering the client copy either by phone or through our on-line registration servers. The types of Licenses available to the vendor/developer are presented below:

   a. **Per Transaction Fee -** Special licensing for applications that are based on a single, or multiple servers that are used to dispense maps, directions, etc. to clients through a wide LAN or web/internet connection. Please, note that you need a special Licensing Agreement for this type of deployment. POIs cannot be part of this type of license.

   b. **Pre-Pay Only:** Each vendor has an escrow account on our registration servers. Money can be deposited into the account by the vendor at any time, by logging to www.registermyapp.com/login.asp and making an on-line payment using their credit card, or by mailing Undertow a check for deposit into the account. When an end user client seat is about to be registered, the escrow account is checked and if there is enough $$ to process the registration, it goes through. If there is not enough $$, a message is displayed in the end-user's registration dialog to contact UnderTow Software. Note that this is the preferred license, in particular if vendors want to pre-purchase a number of licenses so that they can get a discount. It also is preferred because it allows the end-user to register themselves, without any interaction by the vendor being required.

   c. **Pre-Pay Register by Vendor:** Same as option (d) above, but the registration can be done ONLY by the vendor (if vendors require such control), by signing onto the UnderTow registration servers using their vendor code and vendor accounting password.

   d. **Trial Version (Sell it):** This is probably the simplest implementation from the developer/vendor standpoint. The vendor develops and deploys their application and at the time of the client seat registration, either the end user or the vendor can pay for it, at that time, using our on-line purchase system (they will be prompted and be directed to our site's shopping cart).

Why the Need for Registration?

Every copy of your end-user application that uses the OCX and/or the data supplied by UnderTow Software needs to be registered with the UnderTow Software registration servers.

One of the main reasons is that we need to account for all copies of the dataset licensed to end-users, as part of our contractual obligations to our data providers.

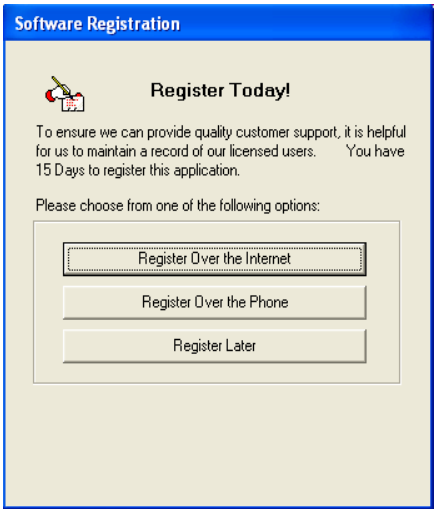
Ways to Register End-User Client Licenses

The registration process may be realized in a number of different ways.

1. The OCX attempts to take care of this registration process without the need for intervention from the developer. It detects when it is in "end-user" mode and the first time it is instantiated in the end-user's environment, it starts a timer the gives the end-user 15 days to register. It also displays a registration dialog, that reminds the user of the need to register and gives them an option to :

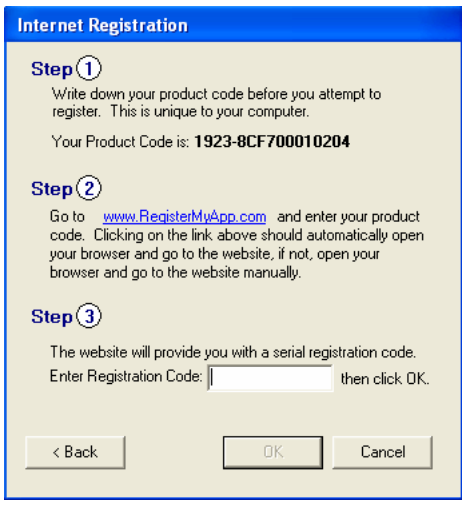
   a. Register Over the Internet, or
   b. Register over the Phone, or
   c. Register later (within the 15 days)

   It should be pointed out that regardless of the registration option selected, or the time of registration, one piece of information is always required, the **Product Code**. It is a number unique to the computer the product is installed on, and it is displayed for the user in the next step of the registration process.



Note: The days left to register the end user application is displayed, as well.

If ***Register Over the Internet*** is selected, then the following dialog is presented.

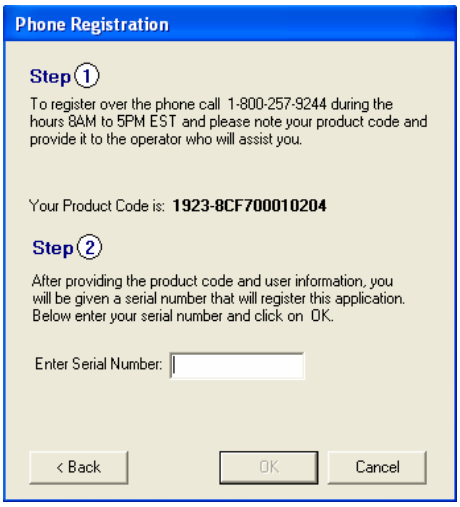


The dialog clearly indicates the 3 steps that need to be takes to complete the registration over the Internet.

Note that the **Product Number**, which will be asked of the user when logging onto www.registermyapp.com, is prominently displayed in this dialog.

After the Serial Registration Code is obtained, the user enters it in the appropriate area in the dialog and clicks O.K.

If ***Register Over the Phone*** is selected, then the following dialog is presented.



The dialog clearly indicates the 2 steps that need to be takes to complete the registration over the Internet.

Note that the **Product Number**, which will be asked of the user when calling the indicated telephone number, is prominently displayed in this dialog.

After the Serial Registration Code is obtained, the user enters it in the appropriate area in the dialog and clicks O.K.

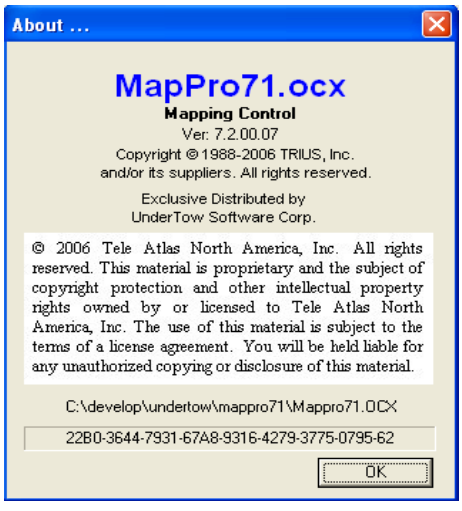
The Registration is complete.

If ***Register Later*** is selected, then the dialog disappears and does not come up until the next time the application is started. It is important to remember, however, that after the 15 day evaluation, the registration dialog that appears has the Register Later option disabled, and the only thing the user can do is register either over the phone, or over the Internet.

If the user exits the dialog, it will reopen every time the user attempts any map manipulation, until such time when they register the application.

2. If, for some reason, the automatic registration process does not fit the requirements of the developer's application, the developer can invoke the registration dialog at will, by calling the routine *ExecRegister(0)*. If the application is already registered, then calling this routine has no affect at all, otherwise the standard registration dialog, displayed above, is invoked. This may be useful, for example, when developing applications that do not use the surface of the OCX to paint the map, but a user-defined control surface, instead, in which case the dialog cannot be displayed on its own, and the developer has to make provisions for invoking the dialog themselves. Note that even in this specialized case, the application will stop working if the 15 day grace period is exceeded.

3. A third way to initiate the registration process and invoke the registration dialog is though the About box. While the application is not registered, when the About box is displayed, the number of days left before registration is required, is also displayed. During that period, placing the cursor on the text "*Mapping Control*" and pressing Ctrl-Click, invokes the registration dialog. If the product is already registered, then the *"Evaluation Mode, Days Left"* is not displayed, and pressing Ctrl-Click has no effect. Again, this is useful in situations where the automatic invocation of the registration dialog is not possible, due to special requirements and conditions of the developed application.



* In order to further facilitate developer special requirements, another helper function has been implemented, GetProductCode(), which returns the same product code that would be displayed in the registration dialog. Note that this function returns a valid product code, only if the application is not already registered.