cGPSmapper User Manual

First Published Date: 2005-04-01
Version: 2.4.5
Published Date: 2009-07-16
Total Page Count: 125

1 Contents

2 Introduction

The latest version of this document can be found at http://www.cgpsmapper.com/.
Feel free to e-mail your comments / contributions to the present document to manual@cgpsmapper.com.

2.1 Purpose of this document

This manual explains how to create *vector maps* and then upload them to your Garmin® GPS receiver (or see them in the MapSource software), utilizing the *cGPSmapper* / *sendmap* software.

2.2 Basic Concepts

2.2.1 What is Polish Format (PFM)?

Polish Format is a convenient, text based, format used for saving map information on a computer and transferring map information between computer programs.
Polish format map files cannot be sent directly to a GPS unit. First they must be converted into a format which is understandable to your GPS receiver. A program which performs this conversion is called a "map compiler".

2.2.2 What is cGPSmapper?

cGPSmapper is a command line program which "compiles" files in polish format (*PFM*) and produces a *vector map* in file(s) of a format understandable by your GPS receiver and Garmin® MapSource.

There are different *cGPSmapper* versions (refer to section 12.3.4 on page 117) with diverse features.

2.2.3 What is sendmap?

sendmap is a command line program used to transfer *vector map* files (generated with *cGPSmapper*) to your GPS receiver.

2.3 Document Conventions

Text in *italics* is shown in the Glossary (page 95).

2.3.1 PFM Code

Text in `monospace` font represents literals (to be inserted literally in the *PFM* file).
Text in `underlined monospace` font represents metavariables - which appear to the right of the equals sign (=) in many statements. Metavariables should be replaced with appropriate values, as described in the explanation (or self-evident).
Text in normal font is explanatory and should not be inserted into the source file.
The number sign special metavariable (`#`) takes a numeric value. E.g. `Data#` stands for `Data0`, `Data1`, etc.
The special iteration operator ... in a statement line has its intuitive meaning. This operator in a separate line denotes that the preceding statement may be repeated zero or more times

with various (typically consecutive) values of the metavariable #. If the iteration operator is preceded by a pair of statements with # metavariables, the whole pair should be repeated (see specific statements for examples).
Text in orange colour (e.g. `Name=`) are mandatory statements in the given section. Text in olive colour (e.g. `Label=`) are optional statements.

2.3.2 cGPSmapper versions

The table below contains the meaning of the different symbols used in this document to represent the cGPSmapper Version to which a certain concept applies.

Symbol	cGPSmapper Version
ϕ	Freeware
σ	Shareware
π	Pro
₪	Routable

☞ The different *cGPSmapper* versions are explained in section 12.3.4, on page 117.

2.4 Manual Authors

This manual was written by Stanislaw Kozicki (the author of cGPSmapper), Gary Turner, Graham Bowring, Hans Scheffler, Keith Sheppard, Greg Rikker and Mauricio Zalba.

3 Overview

Creating a map to be uploaded to a GPS receiver may be compared to programming: you write a program (i.e. a map) in the programming language (i.e. in *PFM*) and then compile it. Alternatively - just as with programming - tools exist to generate the source code visually or semi-automatically or to assist in other ways in the code preparation.
The source code format used by the *cGPSmapper* compiler is referred to as *PFM* (Polski Format Mapy - Polish Map Format) or the "Polish format". The standard file extension for maps in the *PFM* format is **.mp** (in previous versions, the .txt extension was used, which is still acceptable, but not recommended).
A map consists of map objects which fall into four categories: POIs (points of interest, e.g. hotel, restaurant), points (non-indexed point objects, e.g. summit, building), polylines (linear objects, e.g. street, stream), and polygons (area objects, e.g. lake, forest). For non-dimensional objects (POIs and points), it is necessary to define the object attributes, such as label and type, as well as the object coordinate pair (latitude, longitude). For dimensional objects (polylines and polygons), it is necessary to define the object attributes, as well as coordinate pairs of all object vertices. Providing the coordinates is the most laborious part of map authoring.
You may prepare the map source file (.mp) using various methods: by writing the complete source code with any text editor, by generating it visually (by drawing on the screen) with any visual editor, by importing objects (waypoints and tracks) created by the OziExplorer mapping software, or by various combinations of those methods.
When you have finished your map, you can compile it with cGPSmapper (a number of methods are available) and preview it after compilation. The standard file extension for compiled maps is **.img**. Finally, you can upload the resulting compiled map file (.img) to your GPS with *sendmap* or MapSource. All those operations and variants are described in relevant sections below.

4 Map Project

4.1 Map Creation

You write the source file in the *PFM* format (the **.mp** file) using any text editor. All map-related information is provided in relevant statements. Then the map is compiled with the *cGPSmapper* compiler and the resulting .img file is uploaded using *sendmap* or MapSource.
The *PFM* format is described section 4.2 (PFM syntax Description), on page 8.
When you have finished your map (or at any time during the map creation process), you may preview it on the computer screen. Some software packages allow you to preview *PFM* format files directly. Alternatively you can compile it and preview the resultant .img file using MapSource. Finally the .img file may be uploaded to your GPS.

4.2 PFM syntax Description

A *PFM* format file contains comment lines and statements. Blank lines are also permitted
A comment line starts with the ";" character. Comment lines and blank lines may appear at anywhere in the file and are ignored[1] by the compiler.
Statements are grouped into sections. Sections are identified with a section name, enclosed between "[]" and finish with an `[END]` identifier.

> `[END-section_identifier]` can also be used to finish a section. E.g. `[IMG ID]` ... `[END-IMG ID]`, instead of `[IMG ID]` ... `[END]`.

The following types of sections exist:

Section Type	Identifier(s)
Header	`[IMG ID]`

Declarations	`[COUNTRIES]`
	`[REGIONS]`
	`[CITIES]`
	`[CHART INFO]`

Advanced Declarations	`[DICTIONARY]`
	`[BACKGROUND]`
	`[HIGHWAYS]`
	`[ZIPCODES]`
	`[DEFINITIONS]`

[1] However, GPSMapEdit uses special syntax of comments to specify attachments and such comments are interpreted by GPSMapEdit.

Section Type	Identifier(s)
Body (Objects)	[POI] [POLYLINE] [POLYGON] [PLT] [WPT] [DBX] [SHP] [FILE] [RGN10] [RGN20] [RGN40] [RGN80]

The header section is mandatory and must appear as the first section in the source file. All other sections are optional. Declaration and advanced sections (if any) must appear after the header section, in the order specified here. These sections cannot be repeated.
Objects must appear after declaration and advanced sections (if any), may be in any order, and may be repeated as many times as necessary.
The order of statements in the section body (i.e. between the section keyword statement and the [END] statement) is insignificant.

4.2.1 Header

[IMG ID]	Section identifier
ID=########	Unique identifier (up to 8 decimal digits) for the map. May be only written in a decimal format 11000204
Name=map_name	Map name to be displayed in the GPS receiver's Map Info menu. ☞ Refer to section 10.1 (on page 89) for details. 80 characters maximum.
LBLcoding=x	➢ 6 → compressed label coding (smallest maps) ➢ 9 → full-byte (8-bit) coding (supports national characters, depending on the GPS firmware) ➢ 10 → Unicode / MBCS (depending on the GPS firmware) ☞ Default = 6

	`Codepage=xx`	➢ ≠ 0 → full-byte (8-bit) character coding with the specified codepage is used (depending on the GPS firmware) ➢ 0 → single-byte coding Note: Refer to section 0 on page 55 Note: The delimiters for road numbers (🔗 refer to section 4.2.6, on page 31, for details) are different if full-byte coding is used. Note: Special codes are different for 8-bit coding! ☞ Default = 0
	`Datum=xxx`	➢ W84 → WGS-84 ➢ Custom → Custom dx, dy, dz, semiMajorAxis, invFlattening E.g. for (for WGS84): Custom: 0,0,0,6378137.000, 298.257223563 ➢ ≠ W84 & ≠ Custom → 🔗 refer to the `Datum_List.txt` file (in the cGPSmapper directory) for the full list of supported datums ☞ Default = W84
	`Transparent=x`	➢ Y → a full transparent map will be created ➢ N → a transparent map will not be created ➢ S → a semi-transparent map will be created ☞ Default = N When a transparent map is displayed on a GPS unit, features in the unit's basemap will also be visible. If your map is not transparent, it will obscure the basemap when visible. Semi-transparent map is created in a way a usual map is created, but no background object is created. Full transparent map is created in a way that no background object is created and information in IMG is stored that map is transparent. ☞ In some cases – to have a nice overlapping map it is necessary to use semi-transparency instead of full-transparency.
₪	`MG=x`	Lock on road, search for intersection and show next street name active: ➢ Y → Yes ➢ N → no ☞ Default = N
π	`Numbering=x`	Lock on road, show next street name and house numbers along street active: ➢ Y → Yes ➢ N → no ☞ Default = N

₪	`Routing=x`	Lock on road, show next street name active, house numbers along street and routable maps active: ➢ Y → Yes ➢ N → no Note: for routable maps a special data format required! ☞ Default = N
₪	`Lock=x`	Final IMG file will require unlock code specific for each GPS device. ➢ Y → Yes ➢ N → no ☞ Default = N
₪	`ProductCode=x`	Subfamily identifier used for locked IMG files only – value between 1 and 60. Each subfamily can be unlocked with universal unlock code or separately with individual unlock code. ☞ Default = 1
τ π ₪	`CopyRight=xxxxxxx`	Text visible in welcome page of GPS. 🔗 Refer to section 10.1 (on page 89) for details. 80 characters maximum.
	`Elevation=x`	➢ m → metres ➢ f → feet ☞ Default = f
	`POIOnly=x`	No longer used / supported.
σ τ π ₪	`POIIndex=x`	➢ N → objects will be indexed only if POI index info is explicitly provided ➢ Y → all POI objects will automatically be indexed (may be searched by the Find function in the GPS) ☞ Default = N
τ π ₪	`POINumberFirst=x`	➢ N → the house number will be after the street name ➢ Y → the house number will be before the street name ☞ Default = Y
τ π ₪	`POIZipFirst=x`	➢ N → the ZIP code will be after the street name ➢ Y → the ZIP code will be before the street name ☞ Default = Y
σ τ π ₪	`CountryName=country_name` `=country_name`	Should be used in conjunction with `RegionName`. Defines the default region name for automatic city indexing. If not defined, cities will be indexed only if city index info is explicitly provided. 80 characters maximum. ☞ This is a replacement of depreciated key: DefaultCityCountry

σ τ π ₪

RegionName==region_name	Should be used in conjunction with CountryName. Defines the default country name for automatic city indexing. If not defined, cities will be indexed only if city index info is explicitly provided. 80 characters maximum. ☞ This is a replacement of depreciated key: DefaultRegionCountry
TreSize=n	Maximum allowed region size. A higher value increases the allowable region size, but may decrease the map performance; a lower value may increase the map size. Suggested values: ➢ topo maps: 1000-2000 ➢ city (dense streets): 2000-5000 ➢ countryside: 6000-10000
RgnLimit=n	Maximal number of elements in one region. Can be any value between ~50 and 1024 (values less than 50 don't make sense). Recent experiments show that this parameter does not impact map performance and can be set to maximum allowed value: 1024. Suggested value: ➢ 1024
SimplifyLevel=n	Simplify level for Douglas-Peucker simplification algorithm. The higher value, the less simplification is done. It is important to note, that with high value, gridding (limitation coming from the format) might be visible. ☞ Default = 1 Valid range is from 0.1 up to 10

`PreProcess=x`

Kind of pre-processing:

- G → generalization only (faster method, but 'crossroad' nodes might be removed).
  Also the nodes from intersections may be removed.
  Data will be simplified using Douglas-Peucker polyline simplification algorithm which will ensure that the output is not jagged.
- F (or Y) → full generalization + intersection detection for polylines.
  Unnecessary nodes are not removed if there are intersections (this is important for more advanced maps - at intersections, all the intersecting roads have to have nodes or 'find intersection' won't work).
  This is very similar to 'G' with one important exception - all intersection points of the roads are preserved too (even if according to the simplification algorithm these points should be reduced) - this is especially important when we are interested in using 'find intersection' functionality.
  ✍ Intersections are detected only for the most detailed layer, for the others no trace is done.
- P → very similar to option ‘F’ - full generalization + intersection detection for polylines and polygons. Should be used with caution, as process time might be much longer.
  ✍ Intersections for polygons are detected for all the layers, for polylines – only for the most detailed layer.
- N → no generalization and no intersection detection.
  Unnecessary nodes (from the resolution point of view) will be removed automatically.
  There will be no reduction of the 'oversampled' points in the objects - the only reduction of the points will be done because of alignment to the same coordinates.
  This option should be used if input data is prepared separately for each layer - the data for each layer having already been adjusted to the map author's requirements.

Used only if you explicitly provide data for all layers.

☞ Default = F

`Levels=n`	✍ Refer to section 4.4 (on page 42) for details. Number of levels (layers) in the map (at least 2, not more than 10). ➢ 2 ➢ 3 ➢ 4 ➢ 5 ➢ 6 ➢ 7 ➢ 8 ➢ 9 ➢ 10 Note: the last layer must always be empty, e.g. Levels=3 means that two layers only are available for map objects.
`Level#=g`	Grid size for layer # (layer 0 is the most detailed one). ✍ Refer to section 4.4 (on page 42) for details.
`Zoom#=#`	✍ Refer to section 4.4 (on page 42) for details.
`Preview=x`	✍ Refer to section 9.3 (Creating preview map files), on page 85, for details. ➢ N → map designated for use with GPS will be created ➢ Y → map designated for use as preview map for MapSource will be created ☞ Default = N
`AlignMethod`	No longer used / supported.
`BlockSize`	No longer used / supported.
`LevelFill`	No longer used / supported.
`LevelLimit`	No longer used / supported.
`WorldMap`	No longer used / supported.
`DrawPriority=#`	Value between 0 and 31 indicating the priority used by the GPS to draw the map. The highest value – the highest priority is set. Priority for the transparent maps is calculated by adding always 32 to the set priority by DrawPriority value. ☞ Default = 25.
`Marine=x`	Indicates if the map is of marine type. ➢ N or 0 → non-marine map ➢ Y or 1 → marine map ➢ B or 2 → marine map – compatible with G2 devices ➢ X or 3 → non-marine map – compatible with G2 devices – be aware – Garmin does not produce such non-marine maps compatible with G2 devices only ☞ Default = N. ✍ Refer to section 0 (on page 31) for details.

`LeftSideTraffic=x`	Indicates if driving side is right (default) or left. ➢ N or 0 → driving side is right ➢ Y or 1 → driving side is left ☞ Default = N. ✍ Used for routable maps only – for proper display of roundabout icon / other maneuver icons.
`NT=x`	Experimental. Changes storage format to NT-like. This is NOT YET real Garmin NT format. ➢ N or 0 → normal storage format ➢ Y or 1 → Experimental NT format ☞ Default = N.
`[END]`	Section terminator.

4.2.2 Declarations

✍ The DECLARATION elements must be in the order shown herewith.

4.2.2.1 Countries

Although this section is obsolete, it is still supported.

`[COUNTRIES]`	Declares all countries used for city indexing
`Country#=country_name~[0x1d]abbreviation`	Name and abbreviation used to identify country #. The first # must always be one. E.g.: Country1=United States~[0x1d]US 80 characters maximum.
`...`	The statement above can be repeated as needed. # must be in ascending order.
`[END]`	Section terminator

4.2.2.2 Regions

Although this section is obsolete, it is still supported.

`[REGIONS]`	Declares all regions used for city indexing
`Region#=region_name~[0x1d]abbreviation`	Name and abbreviation used to identify region #. The first # must always be one. Subsequent # must be ordered ascending. E.g.: Region1=New York~[0x1d]NY 80 characters maximum.
`CountryIdx#=country_index`	The country_index represents the number in the corresponding `Country#` statement. The first # (`CountryIdx`) must always be one. Subsequent[2] # must be in ascending order. If included, there must be at least 1 region per country. In theory, the limit is 13,107. E.g.: CountryIdx1=1, means that the current region is located in country 1 (right side of the equals sign).
`...`	The statements above can be repeated as needed.

[2] Unlikely, since each region normally is located in a single country.

[END]	Section terminator.

4.2.2.3 Cities

Although this section is obsolete, it is still supported.

[CITIES]	Declares all cities used for indexing
City#=ciy_name	Name used to identify the city #. The first # must always be one. Subsequent # must be in ascending order. E.g.: City1=New York 80 characters maximum.
RegionIdx#=region_index	The region_index represents the number in the corresponding Region# statement. The first # (RegionIdx) must always be one. Subsequent[3] # must be in ascending order. If included, there must be at least 1 city per Region. In theory, the limit is 13,107. E.g.: RegionIdx1=1, means that the current city is located in Region 1 (right side of the equal sign).
...	The statements above can be repeated as needed.
[END]	Section terminator.

4.2.2.4 *Chart Info*

[CHART INFO]	Declarations for marine charts, attached to the 'marine chart' object - which is created automatically as well (similar to the background object) - and also attached to the 'marine border' line. This section should only be present if in the [IMG ID] section, there is a definition Marine=Y
Name=xxx	Chart Name (e.g. La Plata to Nueva Palmira).
Number=xxx	Chart Code (e.g. Gb3561(a)).
Projection=xxx	Chart Projection (e.g. Mercator).
Published=xxx	Place where the chart was published (e.g. United Kingdom).
Scale=###	Map scale (e.g. 1:100000).
DeltaSN=###	Longitude Delta.
DeltaWE=###	Latitude Delta.
IALA=x	IALA system. The areas that use the 'B' system are the Americas, Japan and the Philippines. The remainder of the world uses the 'A' system. ➢ A ➢ B ☞ Default = A
Print=mmyyyy	Paper chart print date. ✍ Note that MapSource will show the day as "01" (the day field is not available in the GPS).

[3] Unlikely, since each city normally is located in a single region.

`Edition=mmyyyy`	Paper chart edition date. ✍ Note that MapSource will show the day as "01" (the day field is not available in the GPS).
`Correction=ddmmyyyy`	Paper chart correction date.
`Text=xxx`	Very long description / information. There could be several `Text` entries in a single object. 16kb maximum (each entry).
`TextFile=file_name`	File containing a very long description / information. There could be several `TextFile` entries in a single object. The path could be either ➢ absolute or ➢ relative to the current directory. For platform portability, it is recommended to use slashes "/" instead of backslashes "\" to separate directories in the path. In Unix, `file_name` is case sensitive. 16kb maximum (each entry).
`ReferenceEllipsoid=###`	Reference Ellipsoid. ➢ 0 → Krassovsky ➢ 1 → Airy ➢ 2 → Modified Airy ➢ 3 → Australian National ➢ 4 → Bessel 1841 ➢ 5 → Bessel 1841 (Namibia) ➢ 6 → Clarke 1866 ➢ 7 → Clarke 1880 ➢ 8 → Everest (Brunei) ➢ 9 → Everest (India 1830) ➢ 10 → Everest (India 1956) ➢ 11 → Everest (W Malaysia 1948) ➢ 12 → Everest (W Malaysia 1969) ➢ 13 → Modified Everest ➢ 14 → Fischer 1960/Mercury ➢ 15 → Modified Fischer 1960 ➢ 16 → Fischer 1968 ➢ 17 → GRS 1967 ➢ 18 → GRS 1980 ➢ 19 → Helmert 1906 ➢ 20 → Hough ➢ 21 → International ➢ 22 → South American 1969 ➢ 23 → WGS-60 ➢ 24 → WGS-66 ➢ 25 → WGS-72 ➢ 26 → WGS-84 ➢ 27 → Unknown

[END]	Section terminator.

4.2.3 Advanced Declarations

✍ The ADVANCED DECLARATIONS elements must be in the order shown herewith.

4.2.3.1 Background

[BACKGROUND]	τ π ₪ Declares a custom shape for the map – another way to define a custom shape for the map is to use a [POLYGON] section (or [RGN80]) as described in section 4.2.4.2 ☞.
Name=file_name	Name of the ESRI file without extension This should be the full or relative path for the ESRI file, without the extension (which should be .shp for files containing ESRI data)
[END]	Section terminator.

4.2.3.2 Dictionary

[DICTIONARY]	
Level#RGNnn=bitmask	☞ Refer to section 4.5 (on page 47) for details. bitmask → mask used to show / hide the objects. # → level on which the bitmask is applied. **nn** → type of object to which the bitmask is applied.
[END]	Section terminator.

4.2.3.3 Highways

♣ This section will be further documented in a future version of this manual.

[HIGHWAYS]	
[END]	Section terminator.

4.2.3.4 ZIP Codes

Although this section is obsolete, it is still supported.

♣ This section will be further documented in a future version of this manual.

[ZIPCODES]	
[END]	Section terminator.

4.2.3.5 Definitions

☞ Refer to section 5.7, on page 55, for further details.

[DEFINITIONS]	
[END]	Section terminator.

4.2.4 Body (Objects)

✍ BODY objects may be specified in any order.

4.2.4.1 Point of Interest

`[POI]`	Point of interest section identifier. `[RGN10]` (meaning point of interest) and `[RGN20]` (meaning point) may be used instead.
`Type=object_type`	Type of element, may be written in hex or decimal or as a name (valid names are defined in file RGNtyps.txt which you can customised to your requirements).

`SubType=object_type`	`SubType` defines the second byte of the `Type` value. The type of element can be defined either by using the `Type` key only or by using the `Type` and `SubType` keys.
	
	Example:
	`Type=0x0211`
	can be also written as:
	`Type=0x02`
	`SubType=0x11`
	Extended / marine types can be stored in 3 bytes as follow:
	
	`Type=0x010203`
	
	What correspond to:
	
	`Type=0x02`
	`SubType=0x03`
	`Marine=Y`

`Marine=x`	Indicates if the object is extended / marine type. ➢ N or 0 → standard type ➢ Y or 1 → extended type ☞ Default = N
`City=x`	Indicates if the POI is a city. Only used if the `[POI]` alias is used. ➢ N or 0 → not a city (instead of `[RGN10]`) ➢ Y or 1 → city (instead of `[RGN20]`) ☞ Default = N
`Label=object_name`	Name of the object to be shown on the map. 80 characters maximum.
`EndLevel=#`	Refer to section 4.4 (on page 42) for details. The coordinates in the lowest numbered `Data#` line apply up to the specified `EndLevel=#`.

`Data#=(lat,lon)`	`Origin#=(lat,lon)` may be used instead. Object data for layer #. ✍ Refer to section 4.4 (on page 42) for details. Coordinates are in degrees, using the datum defined in the header[4] (or default).
`StreetDesc=xxx`	Applies to `[RGN10]` only. Address for the `[RGN10]` object. 80 characters maximum.
`OvernightParking=x`	Applies to `[RGN10]` only. Indicates if 24 hr parking is allowed. ➢ N or 0 → No ➢ Y or 1 → POI at the exit of a highway will have an 'overnight parking' flag. ☞ Default = N
`Highway=xxx`	Applies to `[RGN10]` only. Name of the Highway. This name will be added to the list of available highways, so it can be searched in some GPS devices. Garmin does not support this feature. 80 characters maximum.
`CityName=xxx`	For `[RGN20]` `CityName` has the same meaning as `Label`. If both `Label` and `CityName` are provided, the one which appears later in the file is used. For `[RGN10]` `CityName` is the name of the city to which the object belongs. Can be used only together with keys `RegionName` and `CountryName`. 80 characters maximum.
`RegionName=xxx`	Name of region to which the object belongs. Can be used only together with keys `CityName` and `CountryName`. 80 characters maximum.
`CountryName=xxx`	Name of country to which the object belongs. Can be used only together with keys `RegionName` and `CityName`. 80 characters maximum.
`Zip=xxx`	Object Zip Code. 80 characters maximum.

[4] Refer to section 4.2.1, on page 9, for further details.

Exit#=(type_of_exit_facility),(direction_to_facility),(facilities),(label)	Applies to [RGN10] only. Additional facilities available at the exit. + type_of_exit_facility + direction_to_facility + facilities + label Integer hex or decimal values as indicated on ✍ section 12.2 (Exits), on page 100. 80 characters maximum.
[END]	Section terminator.

4.2.4.2 Polygon

[POLYGON]	Polygon section identifier. [RGN80] may also be used instead. It is used to define lakes, parks, forests, etc.
Type=object_type	✍ Refer to section 4.2.4.1 (on page 19) for details.
SubType=object_type	SubType defines the second byte of the Type value. The type of element can be defined either by using the Type key only or by using the Type and SubType keys. Example: Type=0x0211 can be also written as: Type=0x02 SubType=0x11
Marine=x	Indicates if the object is extended / marine type. ➢ N or 0 → standard type ➢ Y or 1 → extended type ☞ Default = N
Label=object_name	✍ Refer to section 4.2.4.1 (on page 19) for details.
EndLevel=#	✍ Refer to section 4.2.4.1 (on page 19) for details.

`Background=x`	Declare the custom shape of the map. The background object defines the area of the basemap which is covered by this map. It is recommended that background be only used with maps which have irregular boundaries. If there is only one object set as the background, then the `EndLevel` is automatically set to 9. If there is no background object, or more than one, then the `EndLevel` is not changed. "It is a common mistake to use a background object when defining an island. An island is implemented simply as a hole in the containing polygon. Refer to section 10.6 for details. To create a background object in the shape of the island is quite wrong. A background object is not a 'land'. It should only be used to describe the total area covered by your map. Most maps do not require the use of this object at all! The only exception is when you want to create map with an irregular boundary. In which case you should create ONLY ONE BACKGROUND OBJECT which covers the whole map. If you create a lot of background objects - don't be surprised that map is 'strange', slow etc... ➢ N → No ➢ Y → Yes ☞ Default = N
`Data#=(lat1,lon1),(lat2,lon2)...`	`Origin#=(lat1,lon1),(lat2,lon2)` may be used instead. Object data for layer #. Refer to section 4.4 (on page 42) for details. Coordinates are in degrees, using the datum defined in the header[5] (or default). Normally there will be no more than one data# line for each level. The exception is when creating a polygon with holes in it. Holes in polygons are used to represent islands in lakes or seas, clearings in woods etc. Refer to section 10.6 for information on creating polygons with holes.
`[END]`	Section terminator.

[5] Refer to section 4.2.1, on page 9, for further details.

4.2.4.3 Polyline

`[POLYLINE]`	Polyline section identifier. `[RGN40]`may also be used instead. It is used to define linear objects such as streets, streams, etc.
`Type=object_type`	✍ Refer to section 4.2.4.1 (on page 19) for details.

`SubType=object_type`	`SubType` defines the second byte of the `Type` value. The type of element can be defined either by using the `Type` key only or by using the `Type` and `SubType` keys.
	
	Example:
	`Type=0x0211`
	can be also written as:
	`Type=0x02`
	`SubType=0x11`

`Marine=x`	Indicates if the object is extended / marine type. ➢ N or 0 → standard type ➢ Y or 1 → extended type ☞ Default = N
`Label=object_name`	✍ Refer to section 4.2.4.1 (on page 19) for details.
`Label2=object_name`	Secondary name of the object –only applies to roads.
`EndLevel=#`	✍ Refer to section 4.2.4.1 (on page 19) for details.
`Data#=(lat1,lon1),(lat2,lon2)...`	✍ Refer to section 4.2.4.2 (on page 21) for details.
`StreetDesc=xxx`	Street alias or secondary street name. 80 characters maximum.
`DirIndicator=x`	Show direction of the road when selecting intersection in GPS ➢ 0 → No ➢ 1 → Yes ☞ Default = 0
σπ₪ `CityName=xxx@yyy@zzz`	Name of city to which this object belongs. Can be used only together with keys `RegionName` and `CountryName`. 80 characters maximum. Since version 0.97 of cGPSmapper it is possible to assign more than one city as attribute to street – used for search purposes. That can be used for city name and quarter name – to enable search by both names. ✍ Refer to section 6.4 (on page 62) for details.
σπ₪ `RegionName=xxx@yyy@zzz`	Name of region to which this object belongs. Can be used only together with keys `CityName` and `CountryName`. 80 characters maximum.

`CountryName=xxx`	Name of country to which this object belongs. Can be used only together with keys `RegionName` and `CityName`. 80 characters maximum.
`Zip=xxx`	Object Zip Code. 80 characters maximum.
`RoadID=xxx`	
`Numbers#=xxx`	Refer to section 4.2.6 (on page 31) for details.
`Routeparam=xxx`	
`NodID#=xxx`	
`[END]`	Section terminator.

4.2.4.4 Point of Interest from OziExplorer

`[WPT]`	Point of interest section identifier, with data imported from an OziExplorer `.wpt` file. The object labels are derived from the waypoint description field, not from the waypoint name field
`RgnType=object_category`	➢ 0x10 → POI ➢ 0x20 → point
`Type=object_type`	Refer to section 4.2.4.1 (on page 19) for details.
`EndLevel=#`	Refer to section 4.2.4.1 (on page 19) for details.
`File#=file_name`	`.wpt` file from which data will be imported to layer #. The path could be either ➢ absolute or ➢ relative to the current directory. For platform portability, it is recommended to use slashes "/" instead of backslashes "\" to separate directories in the path. In Unix, `file_name` is case sensitive.
`[END]`	Section terminator.

4.2.4.5 Polyline or Polygon from OziExplorer

`[PLT]`	Polygon / Polyline section identifier, with data imported from an OziExplorer `.plt` file.
`RgnType=object_category`	➢ 0x40 → polyline ➢ 0x80 → polygon
`Type=object_type`	Refer to section 4.2.4.1 (on page 19) for details.

`Label=object_name`	☞ Refer to section 4.2.4.1 (on page 19) for details. If the track imported in the [PLT] section contains multiple segments (i.e. objects), all segments will take the same label (name), as defined by the Label statement. However, it is possible to give a different name to each segment. To achieve this, omit the Label statement and specify the names in an additional file, which should have the same name as the .plt file (including the extension) and the additional extension .txt, e.g. Highways.plt.txt). The file must be in the same directory as the .plt file. Each line in this file specifies the name for the corresponding track segment.
`DirIndicator=#`	Direction indicator, only for streets, highways, etc. ➢ 0 → no direction ➢ 1 → the GPS will show direction of the road (calculated internally by GPS) ☞ Default = 0
`EndLevel=#`	☞ Refer to section 4.2.4.1 (on page 19) for details.
`File#=file_name`	`.plt` file from which data will be imported to layer #. The path could be either ➢ absolute or ➢ relative to the current directory. For platform portability, it is recommended to use slashes "/" instead of backslashes "\" to separate directories in the path. In Unix, `file_name` is case sensitive.
`[END]`	Section terminator.

4.2.4.6 Shapes

`[SHP]`	ESRI shape file section identifier.
`name=file_name`	Name of the ESRI files without extension. This should be the full or relative path for the ESRI files, without the extension (which should be .shp for files containing ESRI data).
`Type=xxx`	Type of objects to be imported from the ESRI files ➢ 16 or RGN10 → POI ➢ 32 or RGN20 → cities ➢ 64 or RGN40 → lines ➢ 128 or RGN80 → polygons ➢ 2 or RGN02 → marine polygons ➢ 3 or RGN03 → marine lines ➢ 4 or RGN04 → marine points
`LabelField=field_name`	Name of the field - in the associated .dbf file - from which cGPSmapper will get the `Label` for each object.

`Label2Field=field_name`	Secondary name for roads. Used in cases where you want a road to have a numeric ID and a name. The secondary name of the road (road number if highway for example) - is not visible in the GPS but is used when searching street by name.
`TypeField=field_name`	Name of the field - in the associated .dbf file - from which cGPSmapper will get the `object_type` for each object. The `field_name` field must contain a decimal or hexadecimal value representing the object type. If both `DefaultType` and `TypeField` are specified, an error occurs, but at least one of them must be specified. ☞ Refer to section 4.4 (on page 42) for details on the valid object types.
`SubTypeField=field_name`	Name of the field – in associated .dbf file – from which cGPSmapper will get the second byte of the `object_type` this is an optional field because the `object_type` can be defined using only TypeField
`DirField=field_name`	➢ N or 0 → Hide street direction on crossroads ➢ Y or 1 → Show street direction on crossroads ☞ Default = N
`Level=#`	Level into which objects will be imported.
`EndLevel=#`	☞ Refer to section 4.4 (on page 42) for details. The coordinates from `Level=#` line apply up to the specified `EndLevel=#`.
`DefaultType=object_type`	Decimal or hexadecimal value representing the object type to be applied when the `TypeField` is not specified. If both `DefaultType` and `TypeField` are specified, an error occurs, but at least one of them must be specified. ☞ Refer to section 4.4 (on page 42) for details on the valid object types.
σ τ π ₪ `CityName=field_name`	Name of the field - in the associated .dbf file - from which cGPSmapper will get the `CityName` for each object. Only used for polylines (i.e. when `Type=RGN40` or `Type=64`) and POIs (i.e. when `Type=RGN20` or `Type=32` or `Type=RGN10` or `Type=16`).
σ τ π ₪ `RegionName=field_name`	Name of the field - in the associated .dbf file - from which cGPSmapper will get the `RegionName` for each object. Should not be present if the `DefaultRegionCountry` element is present in the `[IMG ID]` section. Only used for polylines (i.e. when `Type=RGN40` or `Type=64`) and POIs (i.e. when `Type=RGN20` or `Type=32` or `Type=RGN10` or `Type=16`).

σ τ π ₪	CountryName=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the CountryName for each object. Should not be present if the DefaultCityCountry element is present in the [IMG ID] section. Only used for polylines (i.e. when Type=RGN40 or Type=64) and POIs (i.e. when Type=RGN20 or Type=32 or Type=RGN10 or Type=16).
τ π ₪	HouseNumber=field_name	House number written as a string. Used for address search and routing. Only used for POIs (Type=RGN10 or Type=16).
τ π ₪	StreetDesc=field_name	Street name or additional description. Only used for POIs (Type=RGN10 or Type=16).
τ π ₪	PhoneNumber=field_name	Phone number written as a string. Only used for POIs (Type=RGN10 or Type=16).
τ π ₪	Zip=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the Zip for each object. Only used for polylines (i.e. when Type=RGN40 or Type=64) and POIs (Type=RGN10 or Type=16).
₪	RoadID=field_name	Unique ID number for the road. This is internally used by cGPSmapper to maintain routing data creation. Used for routing.
₪	SpeedType=field_name	This attribute defines the maximum allowed speed - it is used mainly for calculating fastest possible route. There are 8 of them: ✓ 7 → 128 km/h ✓ 6 → 108 km/h - Can be adjusted in MapSource ✓ 5 → 93 km/h - Can be adjusted in MapSource ✓ 4 → 72 km/h - Can be adjusted in MapSource ✓ 3 → 56 km/h - Can be adjusted in MapSource ✓ 2 → 40 km/h - Can be adjusted in MapSource ✓ 1 → 20 km/h ✓ 0 → 8 km/h (ferry) Used for routing.
₪	RoadClass=field_name	This attribute defines the importance of the road for routing. It is one of the most important attributes for routing. The lowest importance is 0, the highest is 4. Road class 4 should be used for Major highways and other main roads. Used for routing.

₪	OneWay=field_name	✓ 1 → one way road, where the permitted direction is always from the beginning of the road to the end, considering the digitalisation direction. ✓ -1 → one-way road, opposite to the digitalisation direction. ✓ 0 → two-way road. Used for routing.
₪	Toll=field_name	Defines that it is a toll road.
₪	VehicleE=field_name	✓ 1 → no emergency vehicles allowed on the road.
₪	VehicleD=field_name	✓ 1 → no delivery vehicles allowed on the road.
₪	VehicleC=field_name	✓ 1 → no cars allowed on the road.
₪	VehicleB=field_name	✓ 1 → no buses allowed on the road.
₪	VehicleT=field_name	✓ 1 → no taxis allowed on the road.
₪	VehicleP=field_name	✓ 1 → no pedestrians allowed on the road.
₪	VehicleI=field_name	✓ 1 → no bicycles allowed on the road.
₪	VehicleR=field_name	✓ 1 → no trucks allowed on the road.
	TextFileLines=field_name	Name of the file with long text for very long description of the object
	TextStart=line_number	Starting line number from TextFileLines file to be imported
	TextEnd=line_number	Ending line number from TextFileLines file to be imported
	TextFile=file_name	Text file name to be imported
	Color=field_name	Refer to section 0 (on page 31) for details.
	Style=field_name	Refer to section 0 (on page 31) for details.
	Height=field_name	Refer to section 0 (on page 31) for details.
	Depth=field_name	Refer to section 0 (on page 31) for details.
	DepthUnit=field_name	Refer to section 0 (on page 31) for details.
	HeightUnit=field_name	Refer to section 0 (on page 31) for details.
	Position=field_name	Refer to section 0 (on page 31) for details.
	DepthFlag=field_name	Refer to section 0 (on page 31) for details.
	FoundationColor=field_name	Refer to section 0 (on page 31) for details.
	Light=field_name	Refer to section 0 (on page 31) for details.

LightType=field_name — Refer to section 0 (on page 31) for details.

Note=field_name — Refer to section 0 (on page 31) for details.

LocalDesignator=field_name — Refer to section 0 (on page 31) for details.

InternationalDesignator=field_name — Refer to section 0 (on page 31) for details.

Period=field_name — Refer to section 0 (on page 31) for details.

HeightAboveFoundation=field_name — Refer to section 0 (on page 31) for details.

HeightAboveDatum=field_name — Refer to section 0 (on page 31) for details.

HeightAboveFoundationUnit=field_name — Refer to section 0 (on page 31) for details.

HeightAboveDatumUnit=field_name — Refer to section 0 (on page 31) for details.

LeadingAngle=field_name — Refer to section 0 (on page 31) for details.

Racon=field_name — Refer to section 0 (on page 31) for details.

DoubleLights=field_name	Refer to section 0 (on page 31) for details.
DoubleLightsHorizontal=field_name	Refer to section 0 (on page 31) for details.
FacilityPoint=field_name	Refer to section 0 (on page 31) for details.
[END]	Section terminator.

4.2.4.7 MapDekode file

♣ This section will be further documented in a future version of this manual.

[DBX]	
name=file_name	Name of a MapDekode file (including extension) to be processed in the current compilation.
[END]	Section terminator.

4.2.4.8 File

[FILE]	Lists other *PFM* files to be included in the current compilation.
name=file_name	Name of a *PFM* file (including extension) to be processed in the current compilation. The compiler processes all the objects (and sections) in the specified file as if they were part of the current file. The file included may contain any section but the [IMG ID] section. You may specify either the full path or the path relative to the current directory.
[END]	Section terminator.

4.2.5 Object elevation

By default, the elevation is defined in feet in *PFM*. To define the elevation in metres, the Elevation=m statement should be defined in the header section (refer to section 4.2.1 on page 9). Since this is a global definition, all elevations on a map must be in the same units (either all in feet or all in metres).

Elevation can be specified for POI objects like summit (Type 0x6616) and depth / height points (Types 0x6200 to 0x6600) as well as for polyline objects like land / depth contours (Types=0x20 to 0x25).

Whenever POI label starts with numbers or numbers are placed after ~[0x1f] separator, they are treated as elevation and might be converted (depends of Elevation setting in the header.

The elevation is entered in the label field of the objects. The following code extract defines a height point with elevation of 668 m (assuming `Elevation=m` is defined in the header section):

```
[RGN10]
Type=0x6300
Label=668
Origin0=(-33.93497,18.38925)
[END-RGN10]
```

A minor land contour with elevation of 1080 m can be defined like this:

```
[RGN40]
Type=0x20
Label=1080
Data0=(-33.96727,18.42540),(-33.96725,18.42557),
(-33.96709,18.42600),(-33.96693,18.42624),(-
33.96682,18.42630),
(-33.96662,18.42627),(-33.96646,18.42581),(-
33.96641,18.42557)
[END-RGN40]
```

Text can be combined with the elevation in the label by using the `~[0x1f]` delimiter to indicate the elevation. Example of a summit with 1084 m elevation:

```
[RGN10]
Type=0x6616
Label=Table Mountain~[0x1f]1084
Origin0=(-33.96664,18.42569)
[END-RGN10]
```

Type 0x6616 require a special code separation before the height value – ~[0x1f] – in case – there is no name of the summit – label should looks like:

```
Label=~[0x1f]1084
```

4.2.6 Road numbers

House numbers along the streets can be defined using `NUMBERS#` key within `[RGN40]` declaration. There could be up to 60 definitions of numbers for a single road.

Each definition consist from the `NumbersX` definition where X is increasing value from 1 up to 60

For using house numbering along streets when working with ESRI data please refer to section 5.7, on page 55.

```
[RGN40]
Type=6
Numbers1=0,E,1,9,O,4,20,2999,2999,Warszawa,Mazowieckie,Polska
,Warszawa,Mazowieckie,Polska
Numbers2=3,B,21,40,N,0,0,2999,2999,Warszawa,Mazowieckie,Polsk
a,Warszawa,Mazowieckie,Polska
[END-RGN40]
```

Where –

```
NumbersX=
     [index of point in the polyline – 0 based],
     [left side numbering style],
     [first number on left side],
     [last number on left side],
     [right side numbering style],
     [first number on right side],
     [last number on right side],
     [left side zip code],
     [right side zip code],
     [left side city],
     [left side region],
     [left side country],
     [right side city],
     [right side region],
     [right side country]
```

Some of the information are optional – if no zip code – it can be replaced by '-1', if no city, region and country info – also it can be replaced by '-1'

```
Numbers1=0,E,1,9,O,4,20,-1,-1,-1,-1
```

First '-1' replaces zip code on left side, second – zip code on the right side, then '-1' replaces city/region/country info on left side and the last – on the right side.

This is equivalent of:

```
Numbers1=0,E,1,9,O,4,20
```

Numbering style can be: N,E,O,B – which suits to: **N**one, **E**ven, **O**dd, **B**oth.

4.2.7 Special codes for object labels

It is possible to use some special codes for labialisation of the objects in the map – these special codes are responsible for both – adding graphics around the label and special behaviour of the label.
Special codes are different for 6-bit and 8-bit coding – for which LBLcoding parameter is responsible.

6-bit coding	8-bit coding	Code meaning
~[0x1b]		Used before a letter forces it to be a lower case
~[0x1b2b]		Separation: on the map visible only the second section (when over 1km), with the mouse sees displayed one the word completely, not separated
~[0x1b2c]	~[0x1c]	Separation: on the map visible only the first section (when over 1km), with the mouse sees displayed one the word completely, not separated
~[0x1e]	~[0x1e]	Separation: on the map visible only the second section (when over 1km), with the mouse sees displayed one the word completely, by blank separated

~[0x1f]	~[0x1f]	Separation: on the map visible only the first section (when over 1km), with the mouse sees displayed one the word completely, by blank separated. Normally used as a separator before elevation
		Example:
		`[RGN10]`
		`..`
		`Label=~[0x1f]500`

~[0x2a]	~[0x01]	
		• Interstate symbol
		☞ name can consist only from digits!
		☞ allowed only at beginning of label
		Example:
		`[RGN40]`
		`Label=~[0x2a]11`
		`..`

~[0x2b]	~[0x02]	 `US Highway - shield` ☞ name can consist only from digits! ☞ allowed only at beginning of label
~[0x2c]	~[0x03]	 `US Highway - round symbol` ☞ name can consist only from digits! ☞ allowed only at beginning of label
~[0x2d]	~[0x04]	 `Highway - big` ☞ allowed only at beginning of label
~[0x2e]	~[0x05]	 `Main road - middle` ☞ allowed only at beginning of label
~[0x2f]	~[0x06]	 `Main road - small` ☞ allowed only at beginning of label

4.3 Marine Charts

Marine charts are a special kind of cartography - similar to Garmin's BlueChart - that is used as navigational aid and contains its own set of marine elements.
Although marine charts are generated, managed and compiled in the same way as *ordinary maps*, certain restrictions apply to the attributes of the objects (e.g. extended attributes like CityName, RegionName, StreetDesc are not applicable - marine objects are not searchable). Such restrictions are explained in the present section.

All the attributes listed herewith are meant to be included in one of the following sections:
`[POI]`, `[POLYLINE]`, `[POLYGON]`.
Refer to section 4.2 (on page 8) for details on these sections.

Marine maps cannot be transparent.

Most of marine types can be now used on usual 'land' maps.

`Marine=x`

Indicates if the object is of marine type.
- N or 0 → non-marine object
- Y or 1 → extended / marine object

Default = `Marine` value from the `[IMG ID]` section. If not present, default = N.
This key is no longer necessary if extended 3-bytes type definition is used:

Extended / marine types can be stored in 3 bytes as follow:

```
Type=0x010203
```

What correspond to:

```
Type=0x02
SubType=0x03
Marine=Y
```

Marine objects use many additional special attributes, which are listed below.

Marine objects have only the following keys in common:

`Type=object_type`

Type of element, may be written in hex or decimal or as a name (valid names are defined in file RGNtyps.txt which you can customised to your requirements).

`SubType=object_type`	`SubType` defines the second byte of the `Type` value.
	The type of element can be defined either by using the `Type` key only or by using the `Type` and `SubType` keys.
	
	Example:
	`Type=0x0211`
	can be also written as:
	`Type=0x02`
	`SubType=0x11`

`Label=object_name`	Optional name of the object to be shown on the map. 80 characters maximum.
`Data#=(lat,lon)`	`Origin#=(lat,lon)` may be used instead. Object data for layer #. Refer to section 4.4 (on page 42) for details. Coordinates are in degrees, using the datum defined in the header[6] (or default).
`EndLevel=#`	Refer to section 4.2.4.1 (on page 19) for details.

Marine objects may also have the following keys, depending on the `object_type`:

`Text=xxx`	Very long description / information displayed in the properties windows of the object. There could be several `Text` entries in a single object. Applies only to following marine types: ✓ polygons of types 0x0700 and 0x0704 ✓ points of types 0x0801 and 0x0902 16kb maximum (each entry).
`TextFile=file_name`	File containing a very long description / information displayed in the properties windows of the object. There could be several `TextFile` entries in a single object. The path could be either ➢ absolute or ➢ relative to the current directory. For platform portability, it is recommended to use slashes "/" instead of backslashes "\" to separate directories in the path. In Unix, `file_name` is case sensitive. 16kb maximum (each entry).

[6] Refer to section 4.2.1, on page 9, for further details.

`Color=##`

Object colour.

- 0 → *** COLOR 0X00
- 1 → *** COLOR 0X01
- 2 → *** COLOR 0X02
- 3 → *** COLOR 0X03
- 4 → *** COLOR 0X04
- 5 → *** COLOR 0X05
- 6 → *** COLOR 0X06
- 7 → *** COLOR 0X07
- 8 → *** COLOR 0X08
- 9 → *** COLOR 0X09
- 10 → *** COLOR 0X0A
- 11 → *** COLOR 0X0B
- 12 → *** COLOR 0X0C
- 13 → *** COLOR 0X0D
- 14 → *** COLOR 0X0E
- 15 → *** COLOR 0X0F

Applies only to following marine types:

- ✓ polylines of types 0x04XX to 0x06XX
- ✓ points of types 0x0500 (coloured text)

`Style=##`

Valid values - sum of:
0x00 to 0x03 (basic styles)
0x10 to 0x30 (extended styles)

- 0x00 →
- 0x01 →
- 0x02 →
- 0x03 →
- 0x10 →
- 0x11 →
- 0x12 →
- 0x13 →
- 0x20 →
- 0x21 →
- 0x22 →
- 0x23 →
- 0x30 →
- 0x31 →
- 0x32 →
- 0x33 →

Applies only to following marine types:

- ✓ polylines of types 0x04XX to 0x06XX
- ✓ polyline of type 0x0108 (Recommended Route - valid values are only between 0x00 and 0x03)

`Height=##.#`

Alias name for `Depth`.
Can not be present if `Depth` is present.

`Depth=##.#`

Point depth / height value with one decimal (e.g. 20.1).
The maximum value is 65535, when value is defined as integer; and 6553, when value is defined as real.

Warning:
This value is not displayed if the "Spot Soundings" setting in the GPS receiver is set to "Off". Please note also that this setting affects also some of the objects (mainly soundings): they will not be displayed at all (because they have no visual representation except the depth/height value). This is dangerous for navigation since very important information will be hidden from the chart. There is also the "Spot Soundings" setting in Preferences of MapSource.

Applies only to following marine types (as height / depth):
- ✓ polylines of types 0x0105 to 0x0107, 0x03XX
- ✓ polygon of types 0x0105 to 0x0107, 0x03XX
- ✓ points of types 0x03XX (soundings, building, spot height)
- ✓ points of types 0x04XX (obstruction)

`DepthUnit=x`

Alias name for `HeightUnit`.
Can not be present if `HeightUnit` is present.
Allowed only when either `Depth` or `Height` is present.

`HeightUnit=x`

- ➢ m → metres
- ➢ f → feet

☞ Default = `Elevation` value from the `[IMG ID]` section.
Allowed only when either `Depth` or `Height` is present.

`Position=#`

Position of the obstruction.
- ➢ 0 → unknown
- ➢ 1 → (empty)
- ➢ 2 → doubtful
- ➢ 3 → existence doubtful
- ➢ 4 → approximate
- ➢ 5 → reported

Applies only to following marine types:
- ✓ points of types 0x04XX (obstruction)

`DepthFlag=#`

Depth info of the obstruction.

- 0 → empty (no depth flag)
- 1 → unknown, dangerous for navigation
- 2 → awash at chart datum
- 3 → unknown, safe for navigation
- 4 → unknown

Applies only to following marine types:

✓ points of types 0x04XX (obstruction)

`FoundationColor=#`

Foundation colour.

- 0x00 →  (generic symbol)
- 0x01 →  red
- 0x02 →  green
- 0x03 →  yellow
- 0x04 →  white
- 0x05 →  black
- 0x06 →  black-yellow
- 0x07 →  white-red
- 0x08 →  black-red
- 0x09 →  white-green
- 0x0a →  red-yellow
- 0x0b →  red-green
- 0x0c →  orange
- 0x0d →  black-yellow-black
- 0x0e →  yellow-black
- 0x0f →  yellow-black-yellow
- 0x10 →  red-white
- 0x11 →  green-red-green
- 0x12 →  red-green-red
- 0x13 →  black-red-black
- 0x14 →  yellow-red-yellow
- 0x15 →  green-red
- 0x16 →  black-white
- 0x17 →  white-orange
- 0x18 →  orange-white
- 0x19 →  green-white

Applies only to following marine types:

✓ points of types 0x02XX

`Light=colour`
`Light=(colour,range)`

Definition of light colour and range.
There are several accepted formats:
- Light=2 define colour of the light (valid for types 0x02XX)
- Light=(3,4) colour 3 and range 4 nm

Ranges always in nautical miles.

Light colours:
- 0x00 → unlit
- 0x01 → red
- 0x02 → green
- 0x03 → white
- 0x04 → blue
- 0x05 → yellow
- 0x06 → violet
- 0x07 → amber

Applies only to following marine types:
- ✓ points of types 0x02XX (accepts ONLY colour attribute!)
- ✓ points of types 0x01XX

σ τ π ₪ `Light=(colour,range,angle)...(colour,range,angle)`

Definition of light colour, nominal range and light sectors.
Light = (2,3,10.0), (3,3,205.0) defines colour 2, range 3 nm from angle 10.0 to 205.0 and colour 3, range 3 nm from angle 205.0 to 10.0

The true (geographic) angles should be used (as opposed to magnetic). Angles are arranged clockwise and are given from seaward toward the light. These are bearings of the light as seen from a vessel crossing the sector lines.

Please note that you can change the “Light sectors” setting both in MapSource and in your GPS receiver to suit your needs.

Applies only to following marine types:
- ✓ points of types 0x02XX (accepts ONLY colour attribute!)
- - points of types 0x01XX

`LightType=xxx`

Definition of the light type.
Can be a number (as decimal or hex) or a letter.
If the value is a letter, then the light type is set to 0x0b (Morse code) and letter is used as a Morse code letter.

Hex values:
- 0x00 → unlit
- 0x01 → fixed
- 0x02 → isophase
- 0x03 → flashing
- 0x04 → group flashing
- 0x05 → composite group flashing
- 0x06 → occulting
- 0x07 → group occulting
- 0x08 → composite group occulting
- 0x09 → long flashing
- 0x0a → group long flashing
- 0x0b → Morse letter - *see above*
- 0x0c → quick
- 0x0d → group quick
- 0x0e → group quick and long flashing
- 0x0f → interrupted quick
- 0x10 → very quick
- 0x11 → group very quick
- 0x12 → group very quick and long flashing
- 0x13 → interrupted very quick
- 0x14 → ultra quick
- 0x15 → interrupted ultra quick
- 0x16 → fixed and occulting
- 0x17 → fixed and group occulting
- 0x18 → fixed and isophase
- 0x19 → fixed and flashing
- 0x1a → fixed and group flashing
- 0x1b → fixed and long flashing
- 0x1c → alternating
- 0x1d → alternating occulting
- 0x1e → alternating flashing
- 0x1f → alternating group flashing

Applies only to following marine types:
- ✓ points of types 0x01XX and 0x02XX

`Note=xxx`

Text visible in the properties window of the object.
Applies only to following marine types:
- ✓ points of types 0x01XX and 0x02XX

`LocalDesignator=xxx`	Text visible in the properties window of the object. Applies only to following marine types: ✓ points of types 0x01XX and 0x02XX
`InternationalDesignator=xxx`	Text visible in the properties window of the object. Applies only to following marine types: ✓ points of types 0x01XX and 0x02XX

`Period=xxx`	Period(s) of the light. Can be single value or series of values.
	Examples:
	`Period=2.3`
	`Period=2.3,2.1,2.3,1.0`
	Applies only to following marine types:
	✓ points of types 0x01XX

`HeightAboveFoundation=##`	Height above foundation. Value visible in the properties window of the object. Applies only to following marine types: ✓ points of types 0x01XX
`HeightAboveDatum=##`	Height above datum. Value visible in the properties window of the object. Applies only to following marine types: ✓ points of types 0x01XX
`HeightAboveFoundationUnit=##`	➢ m → metres ➢ f → feet ☞ Default = `Elevation` value from the `[IMG ID]` section. Allowed only when `HeightAboveFoundation` is present.
`HeightAboveDatumUnit=##`	➢ m → metres ➢ f → feet ☞ Default = `Elevation` value from the `[IMG ID]` section. Allowed only when `HeightAboveDatum` is present.
`LeadingAngle=##.#`	Leading angle (in degrees) for the light, value with one decimal. Example: `LeadingAngle=120.1` Applies only to following marine types: ✓ points of types 0x01XX
`Racon=x`	➢ Y → yes ➢ N → no ☞ Default = N Applies only to following marine types: ✓ Points of types 0x01XX

`DoubleLights=x`	➢ Additional info shown in the properties window. Number of light bubbles. Valid values are from 1 to 7. Applies only to following marine types: ✓ Points of types 0x01XX
`DoubleLightsHorizontal=x`	Additional info shown in the properties window. Double lights horizontal / vertical flag. ➢ Y → yes ➢ N → no ☞ Default = N Applies only to following marine types: ✓ points of types 0x01XX
`FacilityPoint=xxx`	Facility point, sum of flags. ➢ 0x000001 → boat ramp ➢ 0x000002 → drinking water ➢ 0x000004 → restrooms ➢ 0x000008 → picnic area ➢ 0x000010 → campground ➢ 0x000020 → marina ➢ 0x000040 → fuel ➢ 0x000080 → marine supply ➢ 0x000100 → bait and tackle ➢ 0x000200 → groceries ➢ 0x000400 → restaurant ➢ 0x000800 → water/electric hook-up ➢ 0x001000 → boat/motor rental ➢ 0x002000 → guide service ➢ 0x004000 → lodging ➢ 0x008000 → dump station ➢ 0x010000 → handicap accessible Applies only to following marine types: ✓ points of types 0x0903

4.4 Levels

4.4.1 Introduction

Every Garmin® GPS with mapping capability gives you the option to zoom in or out on the map page, either displaying a small area in great detail, or a larger area in less detail.
The selection of map objects which it is appropriate for the unit to display is dependent on the zoom level. For example, when you are zoomed in, you would probably want to see individual buildings on your map. As you zoom out, this level of detail would be inappropriate because it would make the map too cluttered.
Using levels in your *PFM* file allows you to dictate the zoom settings at which your map objects will be visible.
Levels also allow you to display map objects in different ways depending on the zoom level. For example, a lake might appear as a region at some zoom levels but a single point at others. This is achieved by creating two map objects to represent your lake - one a region

and the other a single point, and choosing levels for them so that the appropriate one is displayed at each zoom setting.

You may also want to show only the most important objects like main roads and cities at a wide zoom level and include secondary roads - and other objects such as railroads - at more detailed zoom levels.

Figure 1: less detail map example

Figure 2: more detail map example

You can even have different sets of coordinates for the same map object at different zoom levels. The most usual use for this is to include more detail - perhaps showing every bend in a river, when zoomed in and less detail when zoomed out. Normally this will not be necessary though, because cGPSmapper automatically reduces the number of points in higher zoom levels.

4.4.2 Concepts and Terminology

When you zoom in and out on your GPS' map page, a scale line is displayed in the bottom left corner of the map screen[†]. This is annotated with the current scale, such as "800ft" or "2km" or whatever. In a *PFM* file, these zoom levels are identified using an integer value called the *Hardware Zoom Level*. Hardware zoom levels are in the range 1-24. Note that there isn't a precise one-to-one mapping between the hardware zoom levels and all possible device zoom settings. The hardware zoom level is simply a guide.
Hardware zoom level 24 represents the most detailed map levels on the device. Each successively lower zoom level number maps onto twice the map scale of the previous number.
The approximate mapping of hardware zoom levels onto actual device zoom settings is shown in the table below. Note that this is only approximate. The matching of levels to Garmin® display zoom levels is, unfortunately, not an exact science.

Level no.[7]	GPS Equivalent (Metric)	GPS Equivalent (Imperial)
24	Up to 120m	Up to 500ft
23	200m, 300m	800ft to 0.2 miles
22	500m	0.3 miles
21	800m to 1.2km	0.5 miles
20	2km	0.8 miles to 1.2 miles

[†] eTrex series. Other models may vary.
[7] As previously indicated, the map level settings dictate the level at which objects are visible assuming the GPS user has their unit's map detail level set to Normal.

Level no.[7]	GPS Equivalent (Metric)	GPS Equivalent (Imperial)
19	3km	2 miles to 3 miles
18	5km to 8km	5 miles
17	12km	8 miles to 12 miles
16	20km to 30km	20 miles to 30 miles
15	50km	50 miles
14	80km to 120km	80 miles to 120 miles
13	200km to 300km	200 miles to 300 miles
12	500km to 800km	500 miles

The principle of doubling the map scale for each successive zoom number continues beyond level 12 but lower hardware zoom levels are not really useful. Zoom level 12 represents the most "zoomed out" setting for current Garmin® devices.
The *hardware zoom levels* described above are only ever referred to directly in the **[IMG ID]** section of your *PFM* file. The remainder of the file uses a different level numbering scheme called *Map Zoom Levels*.
Map zoom levels are defined by the map author. Any map can use up to ten map zoom levels numbered 0 to 9. If you use fewer than ten map zoom levels you should use consecutive map zoom level numbers starting at zero.
Within your **[IMG ID]** section you specify the number of map zoom levels you will be using with a line of the form **EndLevel=n**. The relationship between your chosen map zoom levels and the hardware zoom levels using a set of **Level*m*=*h*** lines, where ***m*** is the map zoom level and ***h*** is the corresponding hardware zoom level.
For example:

```
[IMG ID]
Levels=4
Level0=23
Level1=21
Level2=20
Level3=17
```

The above extract specifies that the map uses four map zoom levels. Map zoom level zero corresponds to hardware zoom level 23; map zoom level 1 corresponds to hardware zoom level 21 and so on.

> The hardware zoom levels do not need to be consecutive, but each successive map zoom level must correspond to a smaller hardware zoom level number than the previous one.

The settings in our example specify that map objects and coordinates defined as map level zero, will be used at hardware zoom levels 23 and above. Objects defined as map level 1 will be used at hardware zoom levels 21 and 22, and so on.
The highest numbered map zoom level that you define dictates the zoom level at which your map replaces the GPS unit's base map. In our example, the highest numbered map zoom level is 3 and this corresponds to hardware zoom level 17. What this means is that if the GPS device user zooms in to level 17 or higher it will see your uploaded map. At hardware zoom levels 16 and below it will see the base map.
Your highest map zoom level is only used for the purpose of specifying when your map takes over from the base map. You are not allowed to define map objects and coordinates at this level. So, in our example, map zoom levels 0, 1 and 2 are the only ones available for

defining map objects. Map zoom level 3 is only used to dictate when our map replaces the base map.
This means that you must always define one more map zoom level than you actually need for your map objects, and every map definition must therefore include at least two map zoom levels.
Having understood the relationship between map zoom levels and hardware zoom levels you can effectively ignore hardware zoom levels during the map design process. Within the rest of this section, the term *level* should be interpreted as meaning map zoom level unless explicitly stated to the contrary.

4.4.3 Using levels when defining map objects

Here is an extract from a *PFM* file defining the village of Remenham (Berkshire, UK) as a Point of Interest:

```
[RGN10]
Type=3328
Label=Remenham
Data0=(51.551744,-0.889936)
[END]
```

Note that the coordinate definition line starts **Data0=**. The digit following the word **Data** specifies the level at which these coordinates will be used. This definition only specifies coordinates for level zero. That means that the village will only be visible on the user's GPS at zoom level zero.
Suppose we change this to:

```
[RGN10]
Type=3328
Label=Remenham
Data1=(51.551744,-0.889936)
[END]
```

Now we have defined coordinates for level 1 only. That means that the village will be visible only at zoom level 1. If the GPS user zooms out further than that, or if he zooms in closer, the village will not be visible.
Let's say you want the village to be visible at levels zero, one and two. You could write:

```
[RGN10]
Type=3328
Label=Remenham
Data0=(51.551744,-0.889936)
Data1=(51.551744,-0.889936)
Data2=(51.551744,-0.889936)
[END]
```

However there is an easier and better way. The above definition can be abbreviated using an **EndLevel=*n*** line:

```
[RGN10]
Type=3328
Label=Remenham
EndLevel=3
Data0=(51.551744,-0.889936)
[END]
```

What the **EndLevel=*n*** line says is that the coordinates in the highest numbered **Data*n*=** line apply up to the specified **EndLevel=*n***, starting with the level number in the **Data*n*=** line. In our example, **EndLevel=3**, combined with **Data0=** means that the coordinates apply for three consecutive levels commending with level 0 (i.e. levels 0, 1, 2 and 3).

Levels and **LevelsNumber** are "old" equivalents of **EndLevel**.

4.4.3.1 Tailoring shape according to level.

In the previous example, our map object had the same coordinates at all the levels in which it was visible. For single point objects, there's no need for the coordinates to vary. You don't want your village moving around the countryside as the user zooms in or out on your map!

For lines and regions, you may wish to modify the coordinates according to zoom level. Consider the following footpath:

```
[RGN40]
Type=22
Label=
EndLevel=3
Data0=(51.562624,-1.070283),(51.561637,-
1.070592),(51.561272,-1.069878), (51.560059,-1.064277)
[END]
```

This path will be visible, and have exactly the same shape, at levels 0 to 3. Now let's change it slightly:

```
[RGN40]
Type=22
Label=
EndLevel=2
Data0=(51.562624,-1.070283),(51.561637,-
1.070592),(51.561272,-1.069878), (51.560059,-1.064277)
Data1=(51.562624,-1.070283),(51.561272,-
1.069878),(51.560059,-1.064277)
[END]
```

The path is still visible at levels 0 to 3 but the shape changes subtly between level zero (the greatest detail) and level one. At level zero there are four vertices in the line. At levels one and two this reduces to three vertices. What we are actually doing here is specifying that we are only interested in the precise shape of the bend in the path at the highest zoom level. Reducing the detail at higher zoom levels can reduce the size of your digital map, reducing upload times and helping to fit in memory where otherwise it wouldn't.

In practice, you don't normally need to specify reduced detail explicitly in this way because ***cGPSmapper*** automatically detects when there is unnecessary detail for the target zoom

level. ***cGPSmapper*** automatically averages out consecutive points which are too close to be distinguishable on the GPS screen and discards unnecessary points.
Nevertheless it is useful for the map author to understand how it is possible to take explicit control over the shape of map objects at different levels if and when necessary.

4.4.4 Idiosyncrasies of GPS hardware

The foregoing sections have explained how the map author can control the levels at which map objects are displayed. That's the theory. In practice things can be slightly different.

The first thing to note is that the map level settings dictate the level at which objects are visible assuming the GPS user has their unit's map detail level set to *Normal*. In the eTrex series, for example, the map set-up screen offers five choices for detail level: *Most*, *More*, *Normal*, *Less* and *Least*.

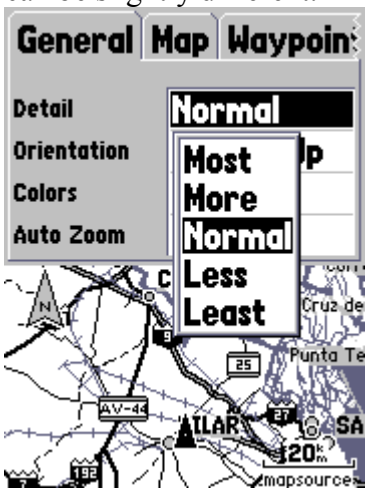


Figure 3: Map Detail Setup

At higher detail levels, map objects will continue to be visible even if the user zooms out further than the greatest level at which you have defined them. At lower detail levels the user will have to zoom in more than expected before the objects become visible. The extent to which the detail setting affects the levels at which objects become visible has not been determined by the authors of this guide. If this is important to you, you will just have to experiment.
Perhaps more bizarrely, the level at which things become visible can also be affected by what type of object it is. You may find, for example, that two points of interest with identical map level settings but different types (for example, one is a building and the other a village) become visible at different zoom settings on the GPS screen. Precise determination of how the object type affects its visibility is beyond the resources of the authors. Again, if this is important to you, you will just have to experiment. As mentioned near the start of this section, choosing correct zoom levels is not a precise science.

4.5 Dictionary

4.5.1 Introduction

Section 4.4 (Levels) explains how you can dictate the zoom settings at which your map objects will be visible. However if you have a large map with many map objects and you need to change the display level for all objects of a specific type, changing the **EndLevel=** tag for each instance can be a cumbersome and time-consuming task.

Fortunately the Polish file format has a solution called 'Dictionary' which allows you to switch map objects of the same type on or off for each level. **[DICTIONARY]** is an optional section and follows after the **[IMG ID]** section.

4.5.2 Concepts and Terminology

[Dictionary] uses strings of 0's and 1's where the position of each digit in the string corresponds to a specific map object type and thus controls the display of that object type. Programmers refer to this as a bit mask. (A bit is a binary digit. In binary there are only two digits, 0 and 1. In a bit mask 0 normally represents 'false' or 'off' and 1 represents 'true' or 'on').

In *PFM* each map object type has a code. Refer to section 12.3 - on page 101 - for a list of object types with codes in hexadecimal and decimal.
You do not have to be familiar with hexadecimal; the decimal equivalent works just as well. (In hexadecimal there are 16 digits, 0 to 9 plus a to f).

In the [Dictionary] 'bit mask' each bit (digit) refers to the object type code corresponding to the bit position, counting from the left of the string. E.g. the first bit refers to map object type 1, the second bit refers to map object type 2 and bit 20 refers to object type 20 and so on.

The *PFM* allows for a 'bit mask' to be defined for each map level.
The format is **Level#RGN*nn*=**<bit mask> where **#** indicates the level, ***nn*** the object class (10, 20, 40 or 80) and **<bit mask>** is a string of 0's and 1's.
If you set the first bit in the 'bit mask' to 0, no objects of type 1 will display on the corresponding level, and if you set the first bit to 1 all objects of type 1 defined for the specific level will display. This is explained much better by an example, see 4.5.3 below

Note that you still need to specify **EndLevel=#** or **Data#=** for each object to extend the object to the required level **#**. The dictionary 'bit mask' only filters out objects on layers they are defined on. However when using dictionary, you can set **EndLevel=** to your highest map level for all objects and control which type of objects display at which map levels with the dictionary 'bit mask'.
Since all object are by default displayed on the most detailed map level, level 0, it is not necessary to define a dictionary 'bit mask' for level 0.
The [Dictionary] tag is optional, but very useful to filter map objects per type per level.

4.5.3 Using Dictionary

The following is an extract from a *PFM* file defining a Dictionary 'bit mask' for polyline [RGN40] objects. In this example major (thick) and principal (medium) highways (types 1, 2 & 3) will be displayed on levels 0 to 3, arterial roads (types 4 & 5) will be displayed on levels 0, 1 and 2 and residential streets (type 6) will be displayed on levels 0 and 1. No other polyline objects will be displayed. Level 4 is the last level of this map and cannot contain map objects.

The first two lines start with ';' indicating that these are comment lines and are ignored by ***cGPSmapper***. They are there so that you do not need to actually count the digits to determine which one refers to which object code.

```
[DICTIONARY]
;           0        1         2         3         4
;           12345678901234567890123456789012345678901234
Level1RGN40=1111110000000000000000000000000000000000000000
Level2RGN40=1111100000000000000000000000000000000000000000
Level3RGN40=1110000000000000000000000000000000000000000000
Level4RGN40=0000000000000000000000000000000000000000000000
[END DICTIONARY]

[RGN40]
Type=0x01
Label=Highway N1
EndLevel=3
Data0=(51.562624,-1.070283), (51.561637,-1.070592),
      (51.561272,-1.069878), (51.560059,-1.064277)
[END]
...
```

In the same way we can use dictionary to filter polygon [RGN80] and point of interest [RGN10 & 20] objects by defining 'bit masks' using **Level#RGN80=** and **Level#RGN10=**. Dictionary for POI [RGN10 and 20] objects works slightly differently in that the POI objects are controlled in groups.
☞ Refer to section 12.3 - on page 101 - for a list of object types with codes in hexadecimal and decimal. Object type codes for POI consist of a group code and a subtype code. For the decimal codes the subtype is shown in brackets after the group code.
The hexadecimal POI object codes are of the form 0x**##nn** where ## is the group code and **nn** the subtype code.
The Dictionary filter operates on the whole group and thus 'bit mask' position ## controls POI group ##. As far as the Dictionary filter is concerned, you can ignore the **nn** subtype code.

The following *PFM* dictionary section will filter out all POI objects and display only dining type POIs 0x2A00 to 0x2AFF on levels 1 to 3.

In this example the comment lines count in hexadecimal, but you can count in decimal if you prefer.

```
[DICTIONARY]
;           0               1               2
;           123456789abcdef0123456789abcdef0123456789abcdef
Level1RGN10=000000000000000000000000000000000000000100000
Level2RGN10=000000000000000000000000000000000000000100000
Level3RGN10=000000000000000000000000000000000000000100000
Level4RGN10=000000000000000000000000000000000000000000000

[END DICTIONARY]
```

Note that the Dictionary 'bit mask' only operates on the objects defined on a layer. To define a specific object on a layer, a **Data#=** entry or an appropriate **EndLevel=** entry is required for the object. Refer to section 4.4 (on page 42) for details.

5 Routable maps

5.1 Introduction

Preparing maps data for making routable maps is a challenge due to additional complexity of the source data comparing to non-routable maps. There are several additional rules that have to be followed in order to have a working map at the end.

The most important thing is to prepare a correct network for roads – assigning route class value to roads does this. Meaning of this attribute is – importance of the road for routing (which usually follow type of the road – but not necessary).
Road network should never be broken – network of most important roads (which are defined by road class attribute) cannot be broken, with lower class attributes insertion – or route calculation will fail.

5.2 General remarks

When preparing routable maps – following rules should be followed:

- IMG file should be created with Level 0 equal to 24 – otherwise distance calculation might be wrong
- A street cannot intersect itself (i.e. a roundabout from one object is not possible as it has common begin with end)
- The minimal distance between Node ID points is 5.4 meters

When preparing routable preview map (what is required when mapset is composed from more than one IMG file – if routing has to work properly under MapSource software)

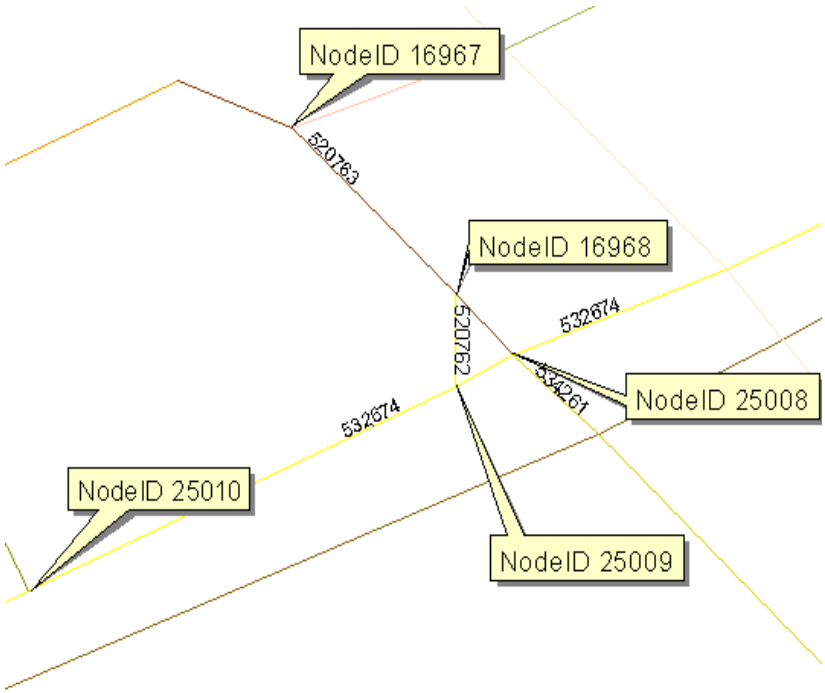
- Roundabouts are not permitted
- Any 2 different streets cannot be connected to each other by both sides
- The minimal distance between NodeID points is 10 meters

Neither CGPSmapper nor cpreview generates automatically routable preview IMG (or MP – in case cpreview) file. Such routable preview map has to be created manually – MP file generated by cpreview program can be used as initial step.

5.3 Concept of input data

The idea of preparing routable data for cGPSmapper base on network of roads – where each road can be defined by its ID number (RoadID) and virtual set of connection points – nod points – where each such node is having its unique ID value (NodeID).

Whenever any two points of roads are sharing same NodeID value – cGPSmapper will make a connection between these roads at given points.



Having above picture, if we define in our data:

RoadID	Node(point)	NodeID
520763	3	25008
532674	2	25008

This will be understood by the program as follow – at point with index 3 of road 520763 (where each point of the road is indexed starting from beginning with index 0) we define NodeID: 25008, then at point with index 2 of road 532674 we define NodeID: 25008 – that means – these roads are connected there – as they are sharing same NodeID value.

To code it in PFM we would look like:

```
[RGN40]
Label=road 1
RoadID=520763
Data0=…
Nod1=3,25008
[END-RGN40]
```

```
[RGN40]
Label=road 2
RoadID=532674
Data0=…
Nod1=2,25008
[END-RGN40]
```

5.4 Road class

The main attribute for routing calculation – as it has been mentioned already – is road class attribute. On one hand it is important to prepare data with non broken road network keeping specific road class, on the other hand it is important to keep well balanced percentage of roads with highest versus other road class roads.

In ideal situation – for usual mixed area (with city and rural areas) – percentage of roads with specific road class should be:

Road class	Approximate number of roads
4	0.5% - 1.5%
3	1% - 3%
2	4% - 7%
1	10%-20%
0	60%-80%

Road Class attribute is used as a main indicator for ‘fastest route’ calculation

5.5 Speed attribute

Second in importance of attributes used for routing calculation is speed attribute – which is used for ‘fastest route’ calculation.

Speed class	Speed attribute	Comment
7	128 km/h	
6	108 km/h	Can be adjusted in MapSource
5	93 km/h	Can be adjusted in MapSource
4	72 km/h	Can be adjusted in MapSource
3	56 km/h	Can be adjusted in MapSource
2	40 km/h	
1	20 km/h	
0	8 km/h	Ferry

This attribute is independent of the route class attribute.

5.6 Routing attributes PMF

Routing attributes of the road in MP

```
[RGN40]
Label=road 1
RoadID=520763
Data0=…
RouteParam=2,3,1
Nod1=3,25008
[END-RGN40]
```

Route attributes are defined by a single key – RouteParam. Each attribute is separated by comma. At lease first two values has to be defined – rest of attributes is optional.

RouteParam=speed,road_class,one_way,toll,denied_emergency,denied_delivery,denied_car,denied_bus,denied_taxi,denied_pedestrain,denied_bicycle,denied_truck

Speed	Speed attribute
Road_class	Road class attribute
One_way	➢ 0 → both directions allowed ➢ 1 → one way road, direction from beginning to end is allowed only ☞ Default = 0
Toll	➢ 0 → normal ➢ 1 → toll road ☞ Default = 0
Denied_emergency	➢ 0 → normal ➢ 1 → emergency car cannot use road ☞ Default = 0
Denied_delivery	➢ 0 → normal ➢ 1 → delivery car cannot use road ☞ Default = 0
Denied_car	➢ 0 → normal ➢ 1 → car cannot use road ☞ Default = 0
Denied_bus	➢ 0 → normal ➢ 1 → bus cannot use road ☞ Default = 0
Denied_taxi	➢ 0 → normal ➢ 1 → taxi cannot use road ☞ Default = 0

Denied_pedestrian	➢ 0 → normal ➢ 1 → pedestrian cannot use road ☞ Default = 0
Denied_bicycle	➢ 0 → normal ➢ 1 → bicycle cannot use road ☞ Default = 0
Denied_truck	➢ 0 → normal ➢ 1 → truck cannot use road ☞ Default = 0

```
For definition of connection – as it was written already –
Nod key is used. For each connection separate key is used :

[RGN40]
Label=road 1
RoadID=520763
Data0=…
RouteParam=2,3,1
Nod1=3,25008
Nod2=4,26005
Nod3=8,99330
[END-RGN40]
```

NodX=point_index,NodeID,bound

Point_index	Index of the point in line definition. Starting point has index 0
NodeID	Index of virtual routing point – NodeID
Bound	➢ 0 → normal NodeID definition ➢ 1 → bound point NodeID definition – this point will be used as entry/exit point of the map – connection to other separate IMG files ☞ Default = 0

Bound attribute defines Entry/Exit points of single IMG file – if no NodeID point with such attribute is defined in the map – there is no way device can enter / exit the map. Typically every road crossing borders of the map, at the border, shall have NodeID point with this attribute set to '1'.

5.7 Routing data defined for ESRI files

When working with ESRI data, routing data can be defined directly for shape files and stored in a DBF file. There can be only one DBF file which defines routing information. This file is used also to define house numbering along streets as well as some other attributes.

Maproute program which is available for free as a generator of routable data by default generates DBF file which is ready to use in the [DEFINITIONS] section of the MP file.

For declaring the MP format [DEFINITIONS] section needs to be used. It is important to note that this section defines only what DBF file should be used and which fields from this particular DBF shall be used to define input data.

Section [DEFINITIONS] declare use of one or two DBF files. First one (declared with field ROUTING) can hold following attributes:
- Routing data
- City / Region / Country attributes
- Zip code attributes

Second DBF file (declared with field RESTRICTIONS) is strictly used for restrictions for routing definitions.

5.7.1 Segment of the road

Routing data – when declared in DBF file – as well as all the additional attributes for roads – are referring to a 'segment of road'. Segment is basically a part of road starting at certain point index of the road (where index 0 refers to a starting point of the road) and ending at the point where next segment starts or at the end of the road.

Let assume – we have a following data in the DBF file :

NodeID	RoadID	Node	LeftStart	LeftEnd
1	100	0	10	12
2	100	2	14	20
	100	5	0	0
3	100	7	22	28

That define first segment of road (road id = 100) starting at the beginning of the road up to point with index 2 – there (as in above example) road numbers will change from 10 (at point index 0) up to 12 (at point index 2). Next segment of this road is starting at point index 2 and ends at point index 5…

As seen in above example, segment definitions does not really needs to be aligned to the routable notes (even if that's most common).

It is not required that data in the input file are sorted in any specific way.

5.7.2 Defining routing data in [DEFINITIONS]

General rule is that each row in the DBF file used in the [DEFINITIONS] section correspond to single NODID point definition.

For a proper NODID point definition – in the input DBF file must exists fields which will at least define :

NODEID	ID of the routable node (Second parameter in the NodX of the MP format NODEID)
ROADID	ID of the road (RoadID in the MP format)
NODE	Index of the point in line definition. Starting point has index 0 (First parameter in the NodX of the MP format)

Optional fields are :

BOUND	Entry / Exit point of the map -

5.7.3 Defining attributes of road segments in [DEFINITIONS]

Same DBF file which define routing data can hold also following additional data – City / Region / Country information for each segment of the road (for left and right side of the road), zip code information and house numbering along the road.

Each segment is defined by the NODE field – which defines at which point of the road segment definition starts. Segment definition is valid till next definition of segment or up to end of the road.

City / Region / Country information definition require definitions of the following fields :

LEFTCITYNAME	Field holding city name on the left side of the segment
RIGHTCITYNAME	Field holding city name on the right side of the segment
LEFTREGIONNAME	Field holding region name on the left side of the segment
RIGHTREGIONNAME	Field holding region name on the right side of the segment
LEFTCOUNTRYNAME	Field holding country name on the left side of the segment
RIGHTCOUNTRYNAME	Field holding country name on the right side of the segment
LEFTZIPNAME	Field holding zip code on the left side of the segment
RIGHTZIPNAME	Field holding zip code on the right side of the segment

5.7.4 Defining restriction data in [DEFINITIONS]

Restriction data needs to be defined in a different DBF file than all the other attributes (routing definition, other road attributes)

Each single restriction is defined in a single row of data in the DBF file. Following fields are required in order to define restriction data:

RESTRICTIONS	Name of the DBF file with restrictions
NODEID1	First NODEID point of restriction
NODEID2	Second NODEID point of restriction
NODEID3	Third NODEID point of restriction

NODEID4	Fourth (only for 4-points restrictions) NODEID point of restriction
ROADID1	First road ID of the restriction
ROADID2	Second road ID of the restriction
ROADID3	Third (only for 4-points restrictions) road ID of the restriction

5.7.5 Secton [DEFINITIONS]

`[DEFINITIONS]`	Routing data, house numbers and restrictions section identifier.
Routing and road attributes part	
`Routing=file_name`	Name of the DBF files with extension for routing and attribute definition. This should be the full or relative path for the DBF file.
`Bound=field_name`	➢ 0 → normal NodeID definition ➢ 1 → bound point NodeID definition – this point will be used as entry/exit point of the map – connection to other separate IMG files ➢ ☞ Default = `0`
`NodeID=field_name`	Name of the field - in the associated .dbf file - from which cGPSmapper will get the `NodeID`
`RoadID=field_name`	Name of the field - in the associated .dbf file - from which cGPSmapper will get the `RoadID`
`Node=field_name`	Name of the field - in the associated .dbf file - from which cGPSmapper will get the `NODE`
`LeftCityName=field_name`	Name of the field - in the associated .dbf file - from which cGPSmapper will get the City name on left side of the road for the current segment
`RightCityName=field_name`	
`LeftRegionName=field_name`	Name of the field - in the associated .dbf file - from which cGPSmapper will get the Region name on left side of the road for the current segment
`RightRegionName=field_name`	
`LeftCountryName=field_name`	Name of the field - in the associated .dbf file - from which cGPSmapper will get the Country name on left side of the road for the current segment
`RightCountryName=field_name`	
`LeftZipName=field_name`	Name of the field - in the associated .dbf file - from which cGPSmapper will get the Zip code on left side of the road for the current segment

RightZipName=field_name	
LeftType=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the numbering type for the left side of the road. This field is optional – as type can be determined from LeftStart and LeftEnd fields. ➢ 0 →No numbers ➢ 1 →Even numbers only ➢ 2 →Odd numbers only ➢ 3 →Even and odd numbers on this side of the street
RightType=field_name	
LeftStart=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the starting number for the left side of the road.
RightStart=field_name	
LeftEnd=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the ending number for the left side of the road.
RightEnd=field_name	
Restrictions part	
Restrictions=file_name	Name of the DBF files with extension for restrictions This should be the full or relative path for the DBF file.
NodeID1=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the first point (NodeID) of the restriction
NodeID2=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the second point (NodeID) of the restriction
NodeID3=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the third point (NodeID) of the restriction
NodeID4=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the fourth (optional – for 4-point restrictions only) point (NodeID) of the restriction
RoadID1=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the first road (Road ID) of the restriction
RoadID2=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the second road (Road ID) of the restriction
RoadID3=field_name	Name of the field - in the associated .dbf file - from which cGPSmapper will get the third (optional – for 4-points restrictions only) road (Road ID) of the restriction

VehicleE=field_name	If from defined field TRUE or larger than 0 value is taken – then restriction apply to emergency vehicles.
VehicleD=field_name	If from defined field TRUE or larger than 0 value is taken – then restriction apply to delivery vehicles
VehicleC=field_name	If from defined field TRUE or larger than 0 value is taken – then restriction apply to cars
VehicleB=field_name	If from defined field TRUE or larger than 0 value is taken – then restriction apply to buses
VehicleT=field_name	If from defined field TRUE or larger than 0 value is taken – then restriction apply to taxis
VehicleP=field_name	If from defined field TRUE or larger than 0 value is taken – then restriction apply to pedestrians
VehicleI=field_name	If from defined field TRUE or larger than 0 value is taken – then restriction apply only to bicycles
VehicleR=field_name	If from defined field TRUE or larger than 0 value is taken – then restriction apply only to trucks

6 National characters - indexing

Garmin map format allows to use several different character coding schema, 6-bit coding which allow to use only basic character set, 8-bit coding and 2 bytes coding. There are however some important constrains when selecting coding schema.

6.1 Codepages

Only 8-bit coding and 2-byte coding allows to use specific codepages. Map format allow to use following codepages:

Codepage	***Name***
936	Simplified Chinese
950	Traditional Chinese
932	Japanes
949	Korean
874	Thai
1252	Western European
1251	Cyrilic
1256	Arabic
1257	Baltic
1250	Central Europe
1253	Greek
1254	Turkish
1255	Hebrew

It is important to know however that the only codepage fully supported by all (or most of) receivers is 1252 – Western European. It is possible to use other codepages – but then national characters may not be shown correctly.

The only fully supported codepage is 1252 – Western European, only with this codepage it is possible to use lowercase characters

Codepage 1252 is the only one that allows using full character set – including lowercase characters. No other codepage is capable to show lowercase characters. While using any other codepage – in the input MP file – all labels should be uppercase – or they will not be correctly presented in receiver.

For every codepage except 1252 All labels must be written in CAPITALS

6.2 Indexing

With certain versions of cGPSmapper it is possible to create indexes that allow searching for certain POI, city or road. With more advanced version it is possible to create global index which collect information from several separate IMG files.

When creating global index, it is possible to index in a simply way (whole labels) or multi-body (if label consist of more than one word, each word is indexed separately).

6-bit coding does not allow to create multi-body indexes.

At the moment cGPSmapper starting from version 0.94 is capable to create maps which handles correctly for indexing only following codepages:

Codepage	***Name***
1252	Western European
1251	Cyrilic
1256	Arabic
1257	Baltic
1250	Central Europe

More codepages will be implemented in the future.

6.3 Preparing maps for indexing

When maps are indented to support 'find' functionality – by city name, region and country – it is very important to use consistent information when preparing source data for compilation.
cGPSmapper starting from version 0.95 during normal IMG file generation is also creating additional IDX file which is used later by cpreview program. IDX file contain index information of objects which is normally lost during conversion from MP format to final IMG format.
After index generation with use of cpreview program, all IDX files can be removed.

It is important to define CODEPAGE in every MP file in the section [IMG ID] – to assure that indexing will work correctly.

By intention there is a special meaning of opening bracket '{' – if first character of street name is such opening bracket – street will not be indexed.
If opening bracket will be not used as a first character – then street will be indexed.
Whenever multi word indexing is used – all words within the label will be indexed except words after opening bracket.

6.4 Multi attributes assignment

Single street can be assigned to more than one city (region) – what is useful whenever we want to make possibility of searching by city name and (for example) by quarter name. Program can handle up to 3 names assigned to a single street.

```
[RGN40]
Label=road 1
CityName=city1@city2
RegionName=region
CountryName=country
Data0=…
[END-RGN40]
```

In above example street will be searchable by both names – `city1` and `city2` – both belonging to `region.`

```
[RGN40]
Label=road 1
CityName=city1@city2@city3
RegionName=region1@region2
CountryName=country
Data0=…
[END-RGN40]
```

Above example shows how to use multi city and multi region configuration. cGPSmapper will assign city1 to region1 and both - city2, city3 to region2.

Depreciated format with CityIdx is supported as well:

```
[RGN40]
Label=road 1
CityIdx=2@3
Data0=…
[END-RGN40]
```

So the road will belongs to both cities – with index 2 and 3.

6.5 Indexing with cpreview program

Indexes for global search are prepared by cpreview external program. Cpreview program accepts same preview control file as one used by cgpsmapper with ‘pv’ command. Cgpsmapper preview generation from now is considered as obsolete – however still available.

As for indexing it is important that a proper sort order – dependant of the codepage – is used, it is important that a proper codepage is defined in the control preview file.

```
[MAP]
…
CodePage=1252
FID=..
ProductCode=..
ID=..
```

The ID key defines map ID value of the preview IMG file. Usually it has to be set to 1. For the meaning of rest of the keys – please see next chapter.

Cpreview program should be used (same as cgpsmapper) from command line – windows shell:

Cpreview pv.txt

Where pv.txt is a control file for preview generation. As the result following files will be generated:

[name].mp
[name].tdb
[name].reg

And – with the version of the program able to generate global indexes:

[name].MDX
[name]_MDR.IMG

Global index file can be generated only with use of the Routable, Personal or Pro version of cGPSmapper.

If option '-r' is used:

Cpreview pv.txt -r

Then first selection when searching for address is for region – instead country.
Refer to chapter on preparing preview files for rest of details.

If option '-m' is used:

Cpreview pv.txt -m

Then each separate word in street name will be indexed. For example – *Albert Einstein* street can be found by 'Albert' and 'Einstein'.

7 Protecting IMG files - locking

7.1 Responsibility

It must be understood that author of the cGPSmapper did not implement locking mechanism by reverse engineering of any of the Garmin original software. All the work has been done basing on 3rd party software not coming originally from Garmin.

THERE IS NO RESPONSIBILITY ON THE AUTHOR SIDE FOR USING THE LOCK TECHNOLOGY – END USER DECIDE ON OWN RESPONSIBILITY TO USE IT KNOWING THAT THIS TOOL WAS NOT APPROVED BY GARMIN AND GARMIN MAY NOT ACCEPT SITUATION THAT 3RD PARTIES ARE CREATING LOCKED MAPS.

7.2 Technology

Starting from the version 0.91 of the cGPSmapper it is possible to prepare locked IMG files. Such files can be used with GPS device only with unique unlock key generated exclusively for the given device. Special registration of cGPSmapper is required.

Locked IMG contains some portion of the data crypt – this data is used by the device to decide whether the end user pose rights to use the map or not. Locking base on using several user defined values – some of them cannot be changed while some of them can.

Each IMG file is locked using 3 values:

FID
- Family ID value – unique identifier for map creator. FID value is hard-coded into the cGPSmapper registration – to ensure that each map creator is using unique value. Some registrations are holding range of allowed FID values – then this value has to be defined in the header of map.

Product Code
- Unique value for product identification – maximum allowed value is 60 therefore map creator can create up to 60 separate products.

Region Code
- Each product can consist from several separate regions. There could be up to 255 regions within single product, each region can have its own unlock code as well as it is possible to generate universal unlock key for every region within a product. It is worth to note that single map set can be build from several different regions – allowing to unlock only specific part of the whole map.

In order to use map in device – it is necessary to use 4th value – GPS ID of the device. This ensure that each sold copy of the map requires unique unlock key. GPS ID value can be usually found in GPS menu under system entry.

It is not possible to use unlock key generated for different GPS ID. The exception is MapSource – which allow to see the map with any valid unlock key – not important for what GPS ID generated.

Starting from cGPSmapper version 0.94 it is possible to generate unlock keys in 2 forms – basing on GPS ID value or locking map to SD card. For that a special key generator is delivered which is working only on Pocket PC platform.

7.3 Locking maps with cGPSmapper

Locking with cGPSmapper is as easy as adding few more entries into the [IMG ID] section. This is:

```
[IMG ID]
…
Lock=Y
ProductCode=
RegionID=
FID=
…
[END]
```

Lock=Y	this will turn on generation of the locked IMG file
ProductCode=	value within a range of 1 up to 60
RegionID=	value within a range of 1 up to 255
FID=	key valid only with multi-FID registrations

IMG created with such additional keys will be locked – meaning – useless without a proper unlock key.

FID value defines general identification of the whole mapset – it has to be unique within all maps uploaded to GPS / used with MapSource.

ProductCode value defines product ID within one FID. Each product is visible as a separate map in MapSource / GPS – however all maps with same FID are sharing the same global index file (MDR file) – therefore it is very important to build a common index file for all the products within the FID.

RegionID value defines a single region within map – which might be unlocked separately. It means that map can be unlocked only in a part – not the whole map at once. Please note – when preparing unlock key – if RegionID is set to 0 (zero) – then it will unlock all regions in the map at once.

7.4 Creating preview – mapset

As locking introduces several new possibilities of creating products, it is important to understand differences.

As it has been mentioned already – single product (mapset) can contain several different regions. Each region may be unlocked separately or end user can use universal unlock key. To achieve that – there are some changes in the preview control file.

First of all – section [MAP]

```
[MAP]
…
Lock=Y
ProductCode=
FID=
```

Lock=Y
: this will turn on generation of the preview for locked IMG files

ProductCode=
: value within a range of 1 up to 60 – must be consistent with detail IMG files – all IMG files must belongs to same product

FID=
: Unique map creator identification, if value used is inconsistent with the one defined by the cGPSmapper registration – it is automatically fixed

Section [FILES]

```
[Files]
region=1,region1
img=10002011.img
img=10002012.img
img=10002013.img

region=2,region2
img=10003011.img
...
[END-Files]
```

It is important to note one new key – region – which is used to define to which region belong following IMG files. In the example above – files 10002011, 10002012, 10002013 belongs to region ID = 1 with name ‘region1’. File 10003011 belongs to region ID = 2 with region name ‘region2’.

It is important to ensure that region ID and product code values are consistent with IMG files itself – as during creation of the IMG file in the [IMG ID] such values are defined. Failure to do so will result in not working mapsets!

7.5 Unlock codes

Along with cGPSmapper there is another program – cGPSmapperKey – which is used to generate unlock codes for products. Program can work in a batch mode or user enter mode. To check how it can work in a batch mode just start the program and examine the help screen.

To generate unlock key following information needs to be provided:

FID
ProductCode
RegionID
GPSID
Expiration date (if exist)

Note on RegionID – to generate universal unlock code working for every region within a given Poduct Code put '0'.
Note on Expiration date – if unlock code shall not expire – then put '0'

7.6 Unlocking maps on SD card

Second method of unlocking maps is to distribute them on SD memory cards with GMAPSUPP.UNL file.
Whole process of distributing maps should be similar – create locked detailed IMG files, prepare appropriate preview files and at the end – creation of GMAPSUPP.IMG file. This file should be then placed in a GARMIN folder at SD card.
In order to enable the map on this particular card, unlock code specific to the card has to be created (in opposite to unlock code specific to the GPS ID value). For that SDLock tool has to be used. This tool is able to retrieve SD serial ID value and generate GMAPSUPP.UNL file that will unlock map on a particular SD card.

SDLock tool – as an optional part of cGPSmapper package – is a Windows Mobile software running on a PocketPC. This special platform requirement comes from the fact that usual PC is not able to read the serial ID value of SD card – while all PocketPC devices have access to this value (of course when the SD slot is built-in into device).

SD card can be processed with SDLock tool before or after putting on the card GMAPSUPP.IMG file. If GARMIN folder is not yet existing on the card – it will be created automatically by the tool.
In addition to the GMAPSUPP.UNL file – SDLocker will create also id.txt file which contains serial ID value of SD card. This file is only for information.

8 Creating custom type file

cGPSmapper creates custom TYP files when invoked with the 'typ' switch. For example,

```
cGPSmapper.exe typ MyCustomTypes.txt
```

Your custom .TYP file may be combined with .IMG files into GMAPSUPP.IMG for uploading into a compatible Garmin GPSr using SendMap 2.0 v3.3 or later, available at **http://cgpsmapper.com/en/buy.htm**. We believe that any Garmin receiver which works with Garmin's POILoader may have custom type definitions installed with your .IMG file. Installed .TYP files do not affect the rendering of Garmin maps – only GMAPSUPP.IMG files uploaded by SendMap.

The generated .TYP file may also be added to your Windows registry enabling MapSource to display your custom types. All installed mapsets are rendered with your .TYP file. This is useful for quickly testing and evaluating your custom types under development.

8.1 Creating the source file

The custom type input file is a simple text file. At this time we recommend using the .TXT extension so it will open with your default text editor.

[_ID] Section

The `[_ID]` section defines the Family ID (FID) that associates this custom type file with your map file. Product Code should be 1; your FID should match the FID declared in the [MAP] section of your preview source file. This definition is used when viewing your custom types in MapSource.

```
[_id]
ProductCode=1
FID=888
CodePage=1252
[End]
```

[_drawOrder] Section

At a minimum, your input file must define the draw order for ALL polygon types – not just your custom ones. Even if you don't define any custom polygon types in your source file, this section is mandatory. If a polygon type is not defined in the `[_drawOrder]` section, it will not be rendered on your GPSr. If a polygon type is not showing up, check to make sure that it is listed in your `[_drawOrder]` section, and that it has a higher priority number than any other overlapping polygons.

Each statement in the `[_drawOrder]` section includes the hex ID of the defined polygon type and its relative draw order. Higher numbers are rendered later. Therefore, a polygon defined with a priority of 1 will be drawn first, and overwritten by an overlapping polygon defined with a higher number (2-8). Priority numbers are between 1 and 8. For example, in the [_drawOrder] section below, a Shopping center (Type 8, priority 3) will be shown on top of a large urban area (Type 0x01, priority 1).

```
[_drawOrder]
;Type=POLYGON_CODE(HEX),PRIORITY
Type=0x01,1         ; Large urban area >200k
Type=0x02,1         ; Small urban area <200k
Type=0x03,1         ; Rural housing area
Type=0x04,1         ; Military base
Type=0x05,1         ; Parking lot
Type=0x06,1         ; Parking garage
Type=0x07,1         ; Airport
Type=0x08,3         ; Shopping center
Type=0x09,1         ; Marina
Type=0x0a,2         ; University/college
Type=0x0b,2         ; Hospital
Type=0x0c,2         ; Industrial complex
Type=0x0d,2         ; Reservation
Type=0x0e,2         ; Airport runway
Type=0x13,2         ; Building/Man-made area
Type=0x14,2         ; National park
Type=0x15,2         ; National park
Type=0x16,2         ; National park
Type=0x17,3         ; City park
Type=0x18,3         ; Golf course
Type=0x19,3         ; Sports complex
Type=0x1a,4         ; Cemetery
Type=0x1e,2         ; State park
Type=0x1f,2         ; State park
Type=0x20,2         ; State park
Type=0x28,1         ; Sea/Ocean
Type=0x29,1         ; Blue-Unknown
Type=0x32,1         ; Sea
Type=0x3b,1         ; Blue-Unknown
Type=0x3c,8         ; Large lake (250-600 km2)
Type=0x3d,8         ; Large lake (77-250 km2)
Type=0x3e,8         ; Medium lake (25-77 km2)
Type=0x3f,8         ; Medium lake (11-25 km2)
Type=0x40,8         ; Small lake (0.25-11 km2)
Type=0x41,8         ; Small lake (<0.25 km2)
Type=0x42,8         ; Major lake (>3.3tkm2)
Type=0x43,8         ; Major lake (1.1-3.3tkm2)
Type=0x44,4         ; Large lake (0.6-1.1tkm2)
Type=0x45,2         ; Blue-Unknown
Type=0x46,2         ; Major river (>1km)
Type=0x47,2         ; Large river (200m-1km)
Type=0x48,3         ; Medium river (20-200km)
Type=0x49,4         ; Small river (<40m)
Type=0x4c,5         ; Intermittent water
Type=0x4d,5         ; Glacier
```

```
Type=0x4e,5         ; Orchard/plantation
Type=0x4f,5         ; Scrub
Type=0x50,3         ; Forest
Type=0x51,6         ; Wetland/swamp
Type=0x52,4         ; Tundra
Type=0x53,5         ; Sand/tidal/mud flat
[end]
```

8.2 Custom Type Definition

Your custom type definitions will replace the default imagery on your GPSr or Mapsource. All other objects will be rendered with their default imagery.

[_point] Definitions

Points (POIs) define your replacement bitmap for the associated POI type using the XPM format. For example,

```
[_point]
Type=0x01
Dayxpm="16 16 2 1"
"    c None"
"X   c #000000"
"XXXXXXXXXXXXXXXX"
"X              X"
"X              X"
"X              X"
"X              X"
"X              X"
"X              X"
"X              X"
"X              X"
"X              X"
"X              X"
"X              X"
"X              X"
"X              X"
"XXXXXXXXXXXXXXXX"
[end]
```

defines a daytime replacement image for POI type 0x01 (Large city). The rendered image will be a 16 pixel square rectangle with a 1 pixel black border and a transparent interior, as shown in the rendering above.

Extended / marine types can be customized using full, 3-byte type definition : Type=0x010402 (wreck)

In addition, you may also specify up to four language strings defining the default name for the POI category. This is the name displayed when the cursor is over an unlabeled object. For example,

```
string1=0x04,Large city            ; 0x04 = English
string2=0x08,Ciudad grande         ; 0x08 = Spanish
```

defines the string 'Large city' when the GPSr is set for English, and 'Ciudad grande' when the GPSr is set for Spanish. Refer to section 12.3.4 (Custom types name substitution), on page 117 for more information about supported language types.

Point bitmap definitions may be up to 24 x 24 pixels and 254 colors. There may be different definitions for the daytime bitmap and the nighttime bitmap. For nighttime definitions, use `Nightxpm=`. If you do not plan to use `Nightxpm,` it is better to use `xpm=` for a single bitmap definition which will be used in both day and night modes. The first line of the definition describes the bitmap dimensions, number of colors, and number of ASCII characters used to represent each pixel. We will use the following simple definition to describe the individual parts of the definition:

```
[_point]
Type=0x01
Dayxpm="4 4 2 1"
"    c None"
"X   c #000000"
"XXXX"
"X  X"
"X  X"
"XXXX"
[end]
```

`Dayxpm="4 4 2 1"` declares that the definition is 4 pixels wide x 4 pixels tall, with 2 defined colors, and 1 character representing each pixel in the bitmap.

Bitmap colors are defined using hex RGB values. Each color should be declared explicitly – cGPSmapper does not support reserved literals representing standard colors. The only literal allowed is 'None' for transparent pixels.

```
"    c None"        ;Special declaration for transparent
color
"X   c #000000"          ;Black
```

The first character is the ASCII character used to represent the associated color in the bitmap. In this example, we are using a space to represent transparent pixels and an 'X' to represent black pixels. Next is a tab, then the letter 'c' which indicates a color definition, followed by a space, '#', then the hex RGB color value.

Following the color declarations is the bitmap description.

```
"XXXX"
"X  X"
"X  X"
"XXXX"
```

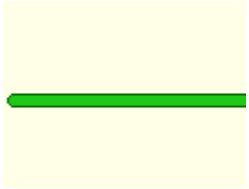
This definition describes a 4x4 rectangle with a black 1-pixel border and a transparent center.

Refer to ↷ section 12.3.5 (How do I create XPM definitions?), on page 118 for a discussion of how to create XPM descriptions using a graphics editor and conversion tools.

[_line] Definitions
Line definitions are used to replace the standard polyline types, including roads. There are two ways to define a line. You may either declare the line's color and thickness attributes for its interior and border, or you may provide a custom bitmap. Both methods allow transparency in the definitions.

Line declaration method 1: Declare a line thickness and border thickness.

```
[_line]
Type=0x01
LineWidth=5
BorderWidth=1
xpm="0 0 4 0"
"1 c #20c818"
"2 c #309838"
"3 c #20c818"
"4 c #086808"
string1=0x04,Toll Road
string2=0x08,Carretera de pago
[end]
```



This definition specifies a replacement for polyline 0x01, Major highway. LineWidth is specified as 5 pixels, BorderWidth is specified as 1 pixel.

```
xpm="0 0 4 0"           ; Define both day and night colors
(4)
```

This line indicates that there is no pixel bitmap, only color definitions. There are 4 colors defined, 2 for daytime, and 2 for nighttime.

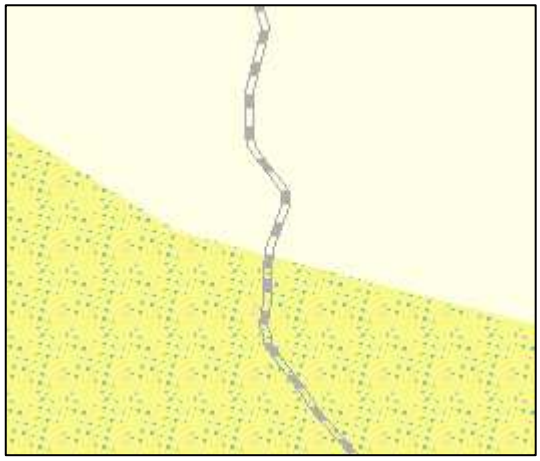
```
"1 c #20c818"           ; Daytime interior color
"2 c #309838"           ; Daytime border color
"3 c #20c818"           ; Nighttime interior color
"4 c #086808"           ; Nighttime border color
```

When describing lines using LineWidth and BorderWidth, note that the color declarations use a different format. The first character represents either daytime interior (1), daytime border (2), nighttime interior (3) or nighttime border (4).

As with POIs and polygons, you may use up to four language substitution strings for the generic type description.

Line declaration method 2: Describe a bitmap using XPM.

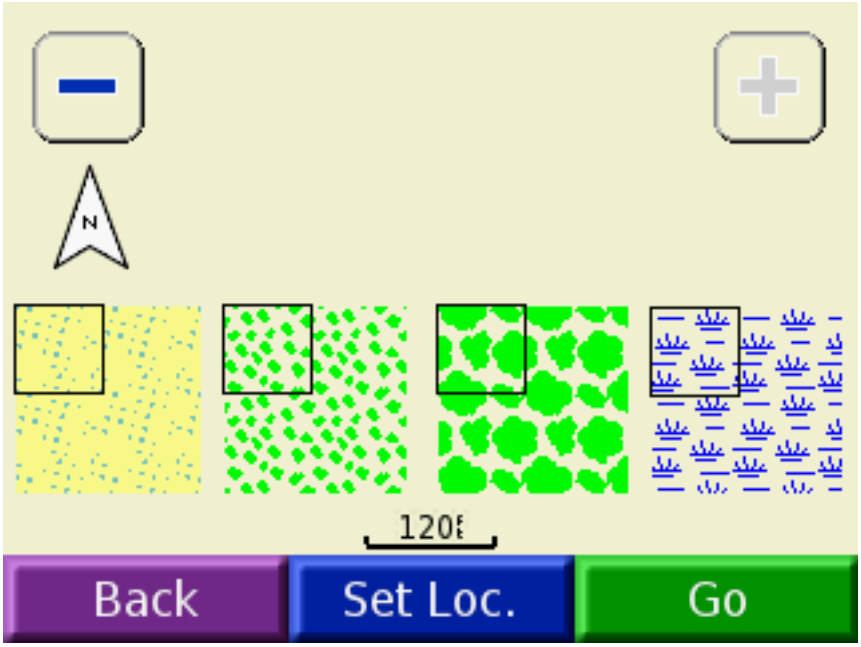
```
[_line]
Type=0x0a
Xpm="32 5 4 1"
"= c #b0b0b0"
"  c none"
"3 c #585858"
"4 c none"
"================================"
"    =======         =======     "
"    =======         =======     "
"    =======         =======     "
"================================"
;12345678901234567890123456789012
string1=0x04,Unpaved
string2=0x08,Camino revistida
[end]
```



The first line of the xpm declaration indicates a definition 32 pixels wide, 5 pixels tall, with 4 colors, using 1 character for the pixel representations in the bitmap. This declaration uses a transparent background, represented by the space character in the ASCII bitmap. In the image above, notice that the transparency reveals whatever texture is underneath the line.

[_polygon] Definitions

Polygon definitions are limited to 32x32 xpm bitmaps using using 2 colors each for the daytime and nighttime definitions. They are tiled when rendered.



In this screen capture from a Garmin nüvi, four polygon types are shown. The one on the left is a custom definition (described below), and the following three are standard type 0x4F, 0x50 and 0x51. The black boxes show 32x32 tiles. Notice that the leftmost tile uses two colors, while the next three tiles use transparency. When defining a polygon bitmap, transparency may be used in either the foreground or background color position.

You may define 2 colors, which will be used for both day and night rendering, or 4 colors with colors 3 and 4 used for nighttime rendering.

```
[_polygon]
Type=0x4e
string1=0x04,Wilderness
string1=0x08,Yermo
xpm="32 32 4 1"
"!   c #faf889"                    ; Daytime foreground
"#   c #77c1c2"                    ; Daytime background
"3   c #77c1c2"                    ; Nighttime foreground
"4   c #faf889"                    ; Nighttime background
"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
"#!!!!!!!!!!!!!!!!!#!!!!!!!!!!!!"
"#!!!!!!!!!!!!!!!!!##!!!!!!!!!!!!"
"!!!!##!!!!!!!!!!!!!!!!!#!!!!!!!"
"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
"!!!!!!!!!!!!!!!#!!!!!!!!!!!!!!!!"
"#!!!!!!!!!!!!!!!##!!!!!!!!!!!!!!"
"!!!##!!!!!!!!!!!!!!!!#!!!!!!!!!"
"!!!!#!!!##!!!!!!!!!!!!!#!!!!#!!!"
"!!!!!!!!##!!!!!!!!!!!!!!!!!!!!!!"
"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
"!!!!!!!!!!!!!!!!!##!!!!!!!!!!!!!"
"!!!!!!!!!!!!!!!!!!!!!##!!!!!!!!!"
"!!!!!!!#!!!!!!!!!!!!!#!!!!#!!!!"
"!!!!!!!!!!##!!!!!!!!!!!!!!!!!!!"
"!!!!!!!!!!!!!!!#!!!!!!!!!!!!!!!!"
"!!!!!!!!!!!!!!!!!!!!#!!!!!!!!!!"
"!!!!!##!!!!!!!!!!!!!##!!!!!!!!!!"
"!!!!!!#!!!##!!!!!!!!!!!!!!!!!!!!"
"!!!!!!!!!!##!!!!!!!!!!!!!!!!!!#!"
"!!!!!!!!!!!!!!!#!!!!!!!!!!!!!!!!"
"##!!!!!!!!!!!!!!!##!!!!!!!!!!!!"
"!!!!##!!!!!!!!!!!!!!#!!!!!!!!!!"
"!!!!!#!!!!!!!!!!!!!!!!!!!!!!#!!"
"!!!!!!!!!!!!!!#!!!!!!!!!!!!!!!!"
"!!!!!!!!!!!!!!!#!!!##!!!!!!!!!!!"
"!!!!!!!!!!!!!!!!!##!!!!!!!!!!!!"
"!!!!!!!!!!!!!!!!!!!!!!#!!!!!!!!"
"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
"!!!!!!!!!!!!!!!!!!##!!!!!!!!!!!!"
"!!#!!!!!!!!!!!!!!!!!#!!!!!!!!!"
[end]
```

8.3 Putting it all together

If you are serious about creating custom types, I recommend that you prepare a small Polish format test file so that you can quickly check your work. Once you have things the way you want them, then you should apply your TYP file to your larger maps.

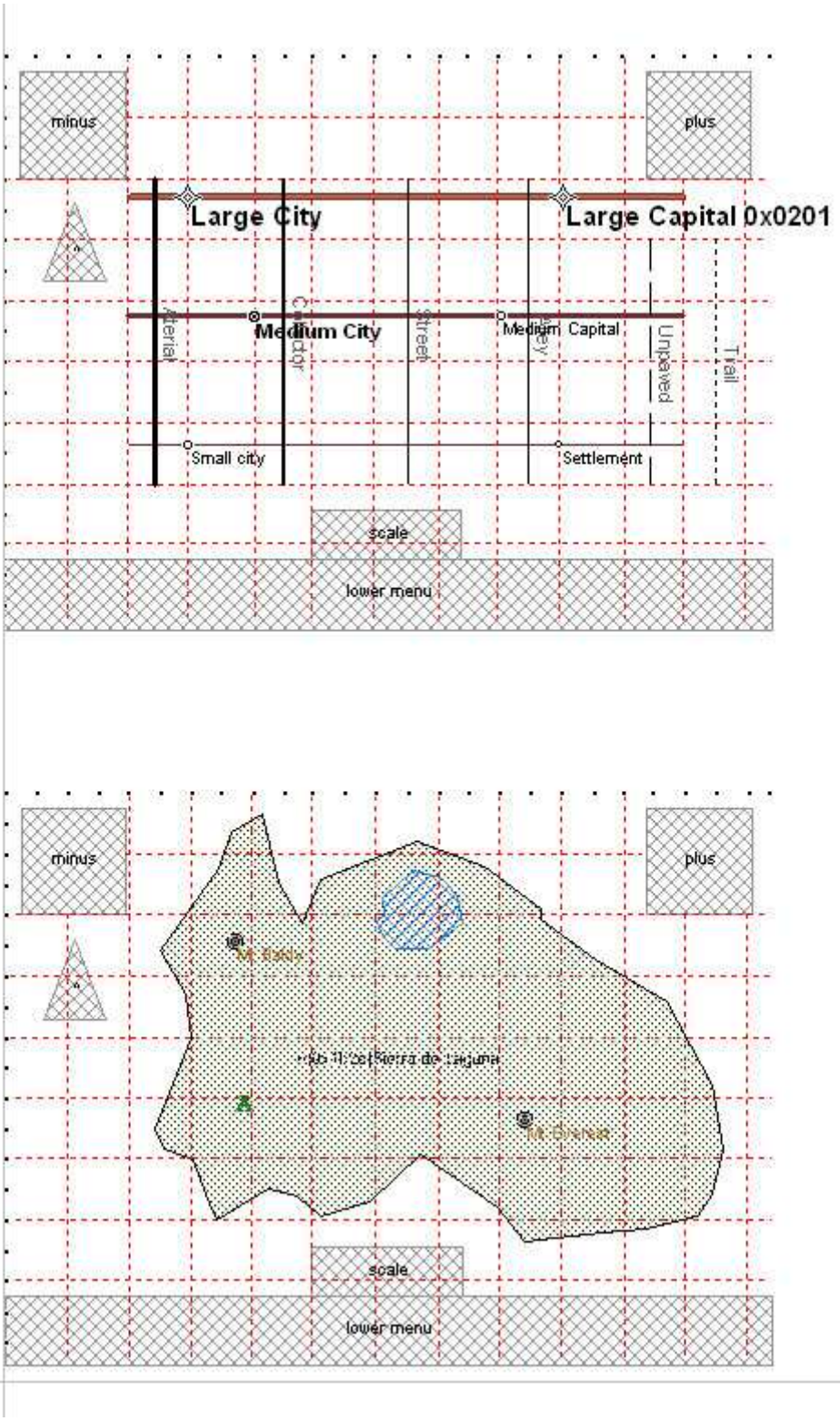
I use MapSource to test my work in progress, as it is much quicker than making a GMAPSUPP.IMG file and downloading it to the unit. Once I'm satisfied with the appearance in MapSource, then I proceed to do the download and inspect the results on the GPSr. Working with small files makes this process much, much easier.

I've also created a template .MP file formatted to fill the nuvi screen exactly. This makes it easy to compose test images that won't be blocked by the menu and on-screen zoom buttons. It's fairly simple to create such a template for your specific unit, and it will save you a lot of time in the long run. I recommend it.

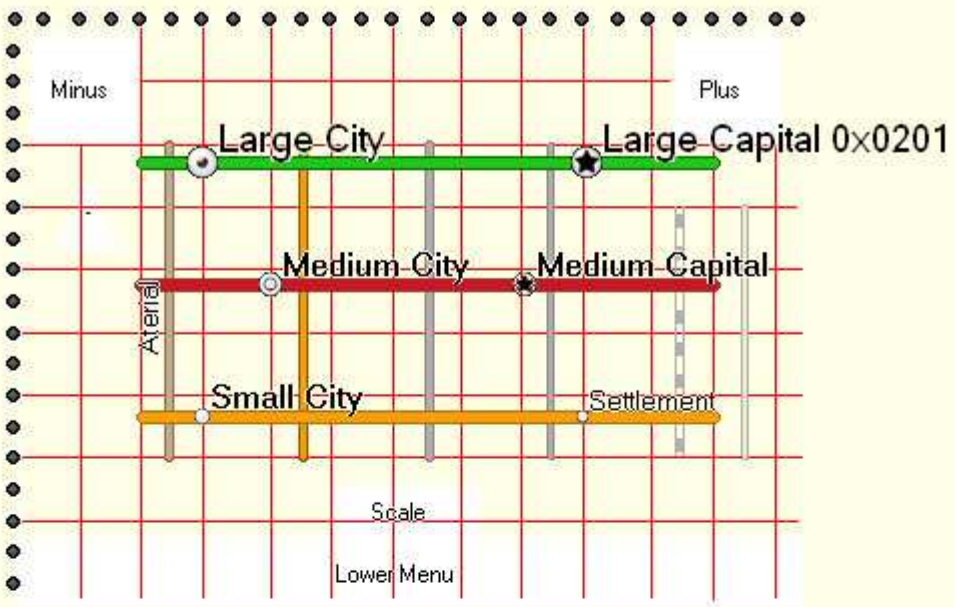
Workflow

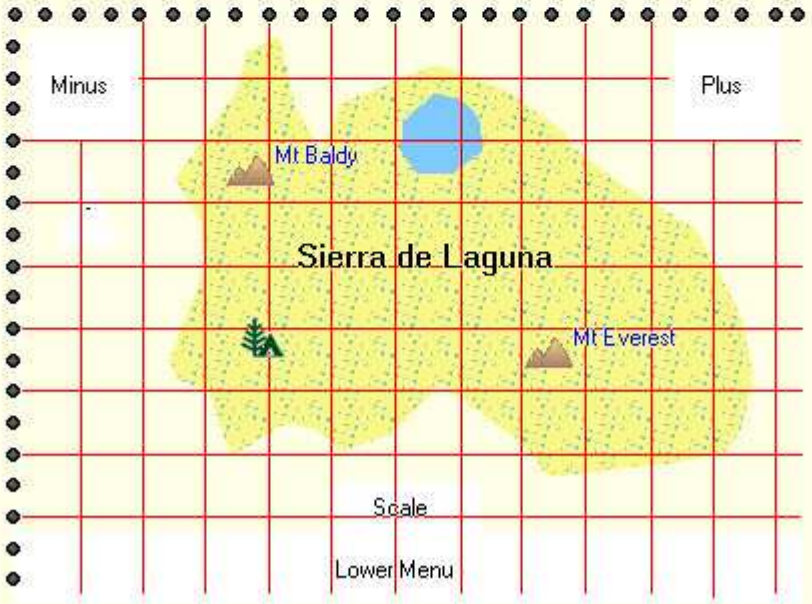
- Create a .MP file with the object types you are customizing. See CustomDemo.mp for an example. Select all of the objects in the file and drag them a location near where your GPSr thinks it is. When you look at your files on the GPSr, you'll only have to drag the map a little bit to see the results. Compile your .MP file.
- Create a .TXT file with your custom type definitions. Use CustomTypesDemo.txt as a starting point to create your own variations.
- Compile your custom type file with cGPSmapper, using the typ switch: cGPSmapper typ CustomTypesDemo.txt
- Use Sendmap 2.0 v 3.3 to assemble your .img file and your .typ file into a single GMAPSUPP.IMG.
- Download GMAPSUPP.IMG into your GPSr. Turn off any loaded mapsets other than your .IMG file to make it easier to find and view your work.
- Repeat.

Sample screens from MapEdit of CustomDemo.mp:



Sample screen from MapSource of CustomDemo.img:

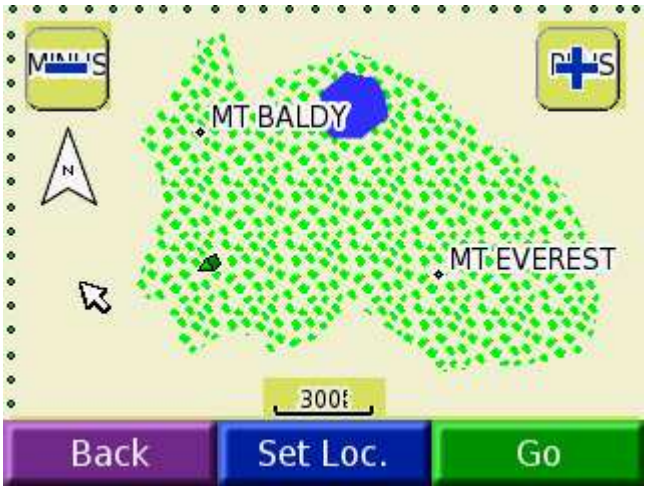




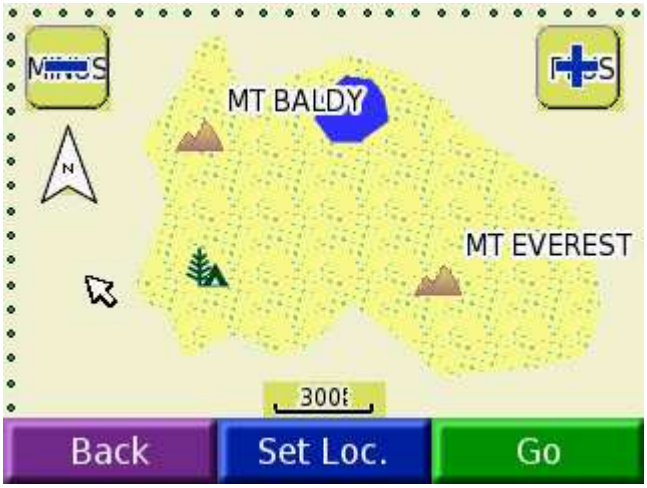
Sample screens from nüvi:

Standard roads and cities

Custom roads and cities



Standard Scrub, camping, mountain range



Custom scrub, camping, mountain range

9 MapSource

MapSource is Garmin's Windows (and only Windows!) PC based GPS interface program. As with most similar programs, it allows upload and download of waypoints, routes and tracks.
Of more interest to us, is the fact that it is also capable of displaying vector-based mapping on the PC, and also uploading the map data to mapping-capable GPS units.

The MapSource program is included with maps bought from Garmin. The same program is shared between the various map sets that Garmin supplies. Recently, Garmin have been supplying MapSource 'Trip and Waypoint manager' free with their mapping GPS units. This program can also be configured to read custom maps.

It was largely the fact that MapSource has to be able to read these maps, that enabled cGPSmapper to be written because the developer was able to use MapSource as a diagnostic tool.

9.1 Installing maps to be used with MapSource

With the most recent version of cGPSmapper sample installation script for Gdansk Routable map is distributed. It requires use of Inno Setup. It simplify installation of map to be used by MapSource to the following steps:

- Build detailed IMG files with use of cGPSmapper
- Build preview files with use of cpreview
- Build preview IMG file from MP file created by cpreview using cGPSmapper
- Prepare installation setup using prepared Inno Setup script
- Distribute map

Below topics are still valid – to be aware of issues when installing maps to be used by MapSource – however in most of the cases use of installation script will be just enough. The only thing to take care is to update FID value in the script as well as names of files to be installed.

9.2 MapSource Data structure

MapSource arranges its map sets as 'products'. Each product - such as 'Metroguide Europe' or 'U.S. Topo' has a top-level 'preview' map, and several/many 'detail' maps. The detail maps can be graphically selected with the program, for upload to the GPS.

Internally, these map sets are configured using data in the Windows registry. Each product requires three registry entries. One entry points at the preview map, one at a 'tdb' file, and one at the location of the detail maps. If you wish to install a custom map set into MapSource, you will need to (a) create the preview map and the tdb file, and (b) create the registry entries to tell MapSource where your files are located.

More recent map products, with routing information - such as Metroguide - are registered in a slightly different way.

9.3 Creating preview map files

To create the preview and tdb file, you will need to re-run cGPSmapper again **after** you have created your .img file. You need to create another text file - similar to a Polish format file, which tells cGPSmapper which detail maps you want to read, and some configuration options. cGPSmapper will then read your detail maps, picking up the map details from the map files, and using these to create the preview map.

```
C:\mymaps> cGPSmapper pv mypv.mp
```

From version 0.94 of cGPSmapper it is recommended to use a separate program to create preview files - cpreview. It uses exactly same input as cGPSmapper for preview generation. It does not however require command 'pv'.

```
C:\mymaps> cpreview mypv.mp
```

In future it is expected that cpreview program will be responsible for global index generation.

The only difference between using cGPSmapper and cpreview solution for preview files generation is a fact that cGPSmapper creates preview IMG file ready to use, while cpreview generates preview map as MP file which then needs to be compile to IMG file. That however allows to easily modify MP file before creating IMG file.

It is required to compile MP file created by cpreview in order to get IMG file. As IMG file used for preview usually is named by name – not by numbers, preferred method of compilation is:

```
C:\mymaps> cgpsmapper mymap.mp
```

9.4 Making the registry entries

To make the registry entries, you can use 3 different methods:-

1. Use the windows registry editor (Start/run/regedit) to directly edit the registry - take **EXTREME** care doing this, as you can completely destroy your PC configuration if you do something silly.
2. Create or edit a registry file, which you can simply double-click to create your entries. You can get a template for this file by doing a registry export. Note that you need double slashes in filenames. If you are distributing your maps to others, this is the simplest way to get them to make the necessary registry changes.
3. Use a GUI program such as MapManager - http://vip.hyperusa.com/~dougs/GPSSM/index.html#GPSMM

The entries are stored in the registry under:

HKEY_LOCAL_MACHINE\SOFTWARE\Garmin\MapSource\Products\##

Where ## is the unique product ID, and **must** correlate with the number you specified with `ProductCode=##` in your mypv.mp file.

The 3 entries are;

- Tdb : the name of the tdb file
- Bmap : the name of the preview img file
- Loc : the directory where the detail img files are stored.

An example registry file:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\Garmin\MapSource\Products\6
67]
"Loc"="D:\\maps\\garmin\\NZ Topo\\"
"Bmap"="D:\\maps\\garmin\\NZ Topo\\NZTopo.img"
"Tdb"="D:\\maps\\garmin\\NZ Topo\\NZTopo.tdb"
```

You only need to do this registration once - for each 'product' that you create. As you edit or create more detail maps, and/or update your preview maps, as long as they stay in the same place on your disk, you do not need to do anything to your registry.

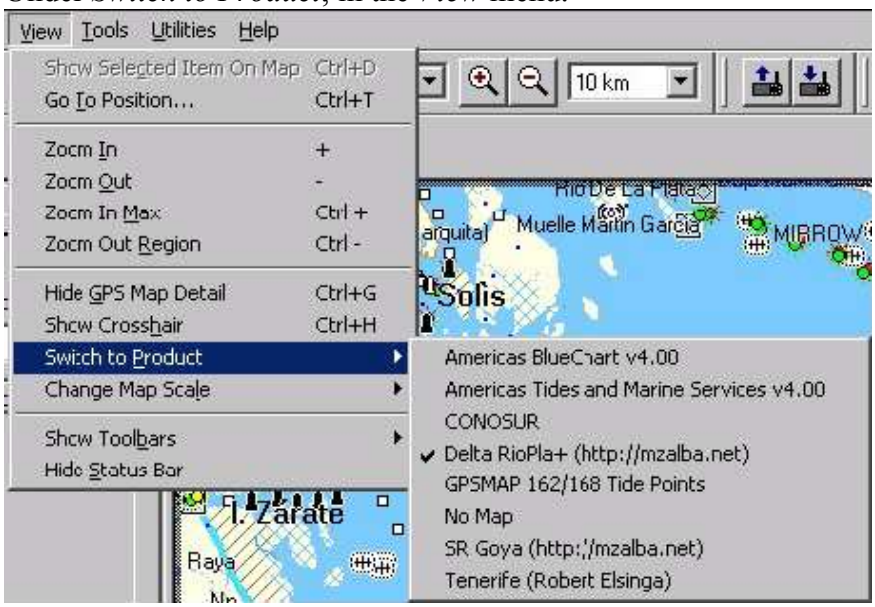
If you plan on distributing your maps, it is NOT a good idea to use the default value - i.e. 66 for the product ID. Rob Mech runs an 'Unofficial Garmin Product ID Database - UGPID' on keenpeople.com - where you can register an ID that hopefully only you will use. Go to:
http://www.keenpeople.com/index.php?option=com_maplist&Itemid=78

9.5 Loading the maps into the GPS

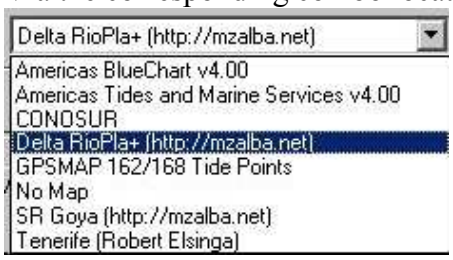
1. Choose the map(s)

Once in MapSource, the map(s) to be loaded in the GPS must be chosen first.
This is done via any of the following methods:

- Under *Switch to Product*, in the *View* menu.



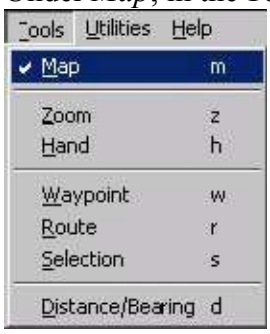
- Via the corresponding combo located in the *View Toolbar*.



The quantity of maps available will vary depending on the maps installed.

2. Select the map(s)
The map(s) to be loaded in the GPS must be selected.
This is done via any of the following methods:

- Under *Map*, in the *Tools* menu.



- Via the respective button located in the *Tools Toolbar*.

Once this is done, the map to be selected must be clicked (in order to select it).
The maps to be transferred to the GPS (and the *bytes* they occupy) will be shown to the left of the screen.

3. Send the map(s) to the GPS
This is done via any of the following methods:

- Under *Send To Device*, in the *Transfers* menu.

- Via the respective button located in the *Transfer Toolbar*.

Once the transfer is done, the program will confirm the map(s) transfer finished successfully.

10 FAQs

10.1 Name variables and where they show up

10.1.1 Introduction

There are three locations where the names of maps, map sets, and related information ("name data") are specified:

- the *PFM* file;
- the *PFM* Preview file; and
- *sendmap* options.

There are various locations where the name data is displayed both on the GPS unit and in the Garmin MapSource software. Figure 4 shows the relationship between where the name data is specified and where it is displayed.

10.1.2 PFM File

The name data in the *PFM* file is used to describe a single *PFM* file as opposed to a collection of *PFM* files. The name data is specified between the `[IMG ID]` tags.

`[IMG ID]`

`Name=map_name`	The name of the map. This field is the first field displayed on the GPS unit under the "Map Information" section. It is displayed in the MapSource software on the "maps" tab when the map is selected and in the Map Properties window. **NOTE:** The name field will not be displayed on the GPS unit if the `ID` field in the *PFM* is not specified as a decimal field or is not listed correctly.

11.2.2 PFM Preview File

The name data in the *PFM* Preview File is used to describe a collection of *PFM* files. The name data is specified between the `[Map]` tags.

`[MAP]`

`MapsourceName=xxxxxx`	The Product name. This field is not displayed on the GPS unit. This field is displayed on the product menu bar and product menu in the MapSource software.
`MapSetName=xxxxxxx`	The Area name. This field is the second field displayed on the GPS unit under the "Map Information" section. It is displayed in the MapSource software on the "maps" tab when the map is selected and in the Map Properties window.
`CDSetName=xxxxxxx`	The CD Set Name. This field is not displayed on the GPS unit. This field is displayed in the MapSource software when displaying the Product Information.

`MapVersion=nnn`	The software version of the CDSetName. This field is not displayed on the GPS unit. This field is displayed in the MapSource software when displaying the Product Information. It will be displayed as n.nn. For example: MapVersion=153 will be displayed as Data Version 1.53 This field can only contain numeric characters and must be three characters long (i.e. 000 through 999).
`Copy1=xxxxxxx`	The first line of the copyright text associated with the CDSetName. This field is not displayed on the GPS unit. This field is displayed in the MapSource software when displaying the Product Information. If you wish to include a copyright symbol ("©") in your text, you can do in your favourite text editor. Hold down the ALT key, type the numbers 0169 on the numeric keypad and then release the ALT key. You must use the numbers on the numeric keypad as opposed to the numbers across the top of the keyboard. You must also have Num Lock turned on.
`Copy2=xxxxxxx`	The second line of the copyright text associated with the CDSetName. See above.
`Copy3=xxxxxxx`	The third line of the copyright text associated with the CDSetName. See above.

11.2.3 Sendmap

The name data specified when using *Sendmap* is used to describe a collection of *PFM* files. The name data is specified as command line options.

Sendmap –M "MapSetName" filename1.img [filename2.img] […]	The Area name. This field is the second field displayed on the GPS unit under the "Map Information" section. *Sendmap* is not used with the MapSource software.

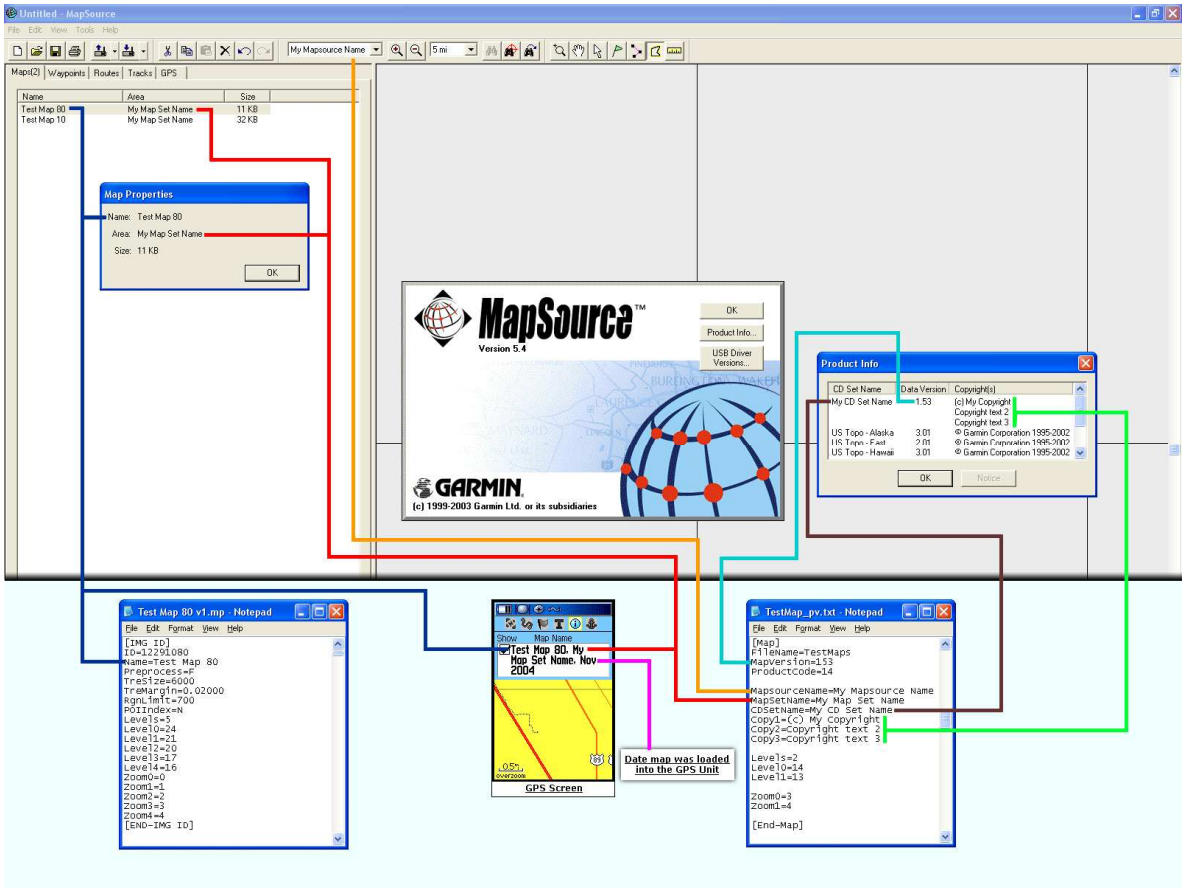


Figure 4: How name variables are shown

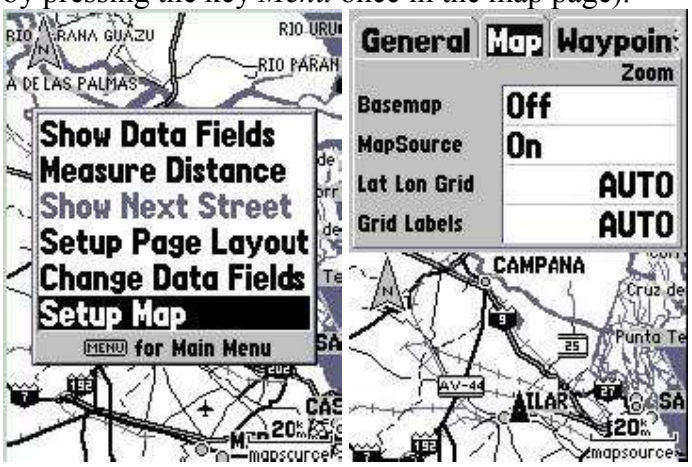
10.2 Activation of maps in the GPS

1. Activation of the map(s) in the GPS

If the map image is not shown in the GPS, check that the map is selected to be shown.

The method for doing this depends on the GPS model. For example you may find it on the *MapSource Info* screen, accessed from the unit's main menu or on the information page of the Setup Map screen. Consult your unit's documentation to find out how to access this screen display.

2. Activation / Deactivation of the GPS' base map
Loaded maps usually have more detail of the zone than the base map (which comes from the factory with the GPS).
Some GPS models allow you to deactivate the base map in the GPS (so that zoom levels do not mix up with the other loaded maps). Depending on the GPS model, this is done via the option *Basemap* in the *Map* tab, in the *Setup Map* menu (which can be accessed by pressing the key *Menu* once in the map page).



Note that you can also use the Transparent= line in your *PFM* file header section.
☞ Refer to section 4.2.4.1 (on page 19) for details.

10.3 Saving Objects as [RGNx0] vs. [POI], [POLYGON], [POLYLINE]

10.3.1 Equivalences

Notation 1	Notation 2
`[POLYLINE]`	`[RGN40]`
`[POLYGON]`	`[RGN80]`

`[POI]`
`City=Y`
`...`
`[END]`
	`[RGN20]`

`[POI]`
`City=N` (or no city key)
`...`
`[END]`
	`[RGN10]`

[POI] covers both [RGN10] and [RGN20]. The difference is made with the 'City=y' key.

10.3.2 Impact of saving objects in one format or the other

There is no impact. [POI], [POLYGON], [POLYLINE] are more understandable to the human reader.

10.3.3 Preferred method

There is no preferred method.

10.4 Relationship between levels in the detail maps and the preview maps

The lowest zoom level in the PV should be the smaller at least by one from the highest in the detailed map.
The preview is displayed up to this switch over point.
When you zoom in more in MapSource the GPS detail map is displayed.

Example
In your detail IMG file:

```
Level0=24
Level1=22
...
Level4=18

Zoom0=0
Zoom1=1
..
Zoom4=4
```

And in the preview file:
;Smaller by one from the top level in the detailed map – which is Level4=18

```
Level0=17
Level1=16

Zoom0=5
Zoom1=6
```

(Does not overlap with detail IMG file)

10.5 Filling (Gas) Stations not showing in the find function of the GPS

Use 0x2F01 (instead of 0x4400) if you want to show it in the find function of your GPS receiver.

10.6 Islands and Clearings

Islands, clearings, etc. are created by defining polygons with "holes" in them. For example, a hole in a polygon representing a lake or the sea will be displayed on your GPS as an island. For this reason, the following technique is often referred to loosely as creating an island. However the same technique may also be used for creating holes in other polygon types. For example, a hole in a wood would represent a clearing and so on.
A hole can be defined in any region by including a second or subsequent `Data#=` line with the same level, or layer, number as the enclosing polygon. The Data# line defining the hole should come after the `Data#=` line for the enclosing polygon and should define a polygon which is wholly contained within the enclosing polygon.
For example, here is a definition of a simple wood containing a clearing:

```
[RGN80]
Type=80
Label=Some Wood
Levels=3
Data0=(52.636651,2.189029),(52.844893,4.709805),(51.465289,5.
707034),(49.799352,4.128087),
(50.033624,2.853849),(51.283077,1.524209)
Data0=(51.595440,2.604541),(52.272227,3.961882),(50.762472,4.
405095),(50.710411,3.906480)
[END]
```

11 Glossary

♣ This section will be further documented in a future version of this manual.

Term	Definition
Vector Map	
cGPSmapper	Map compiler.
sendmap	
PFM	Polish Format is a convenient, text based, format used for saving map information on a computer and transferring map information between computer programs. Polish format map files cannot be sent directly to a GPS unit. First they must be converted into a format which is understandable to your GPS receiver. A program that performs this conversion is called a "map compiler".

12 Appendices

12.1 cGPSmapper compilation Errors and Warnings

The table below contains a list of the different errors and warnings that could occur at compilation time. Codes starting with either "E" or "R" apply only to routable maps. Codes starting with "W" are warnings. Although the compilation will stop after an error occurs, it will not stop when a warning message occurs.

♣ This section will be further documented in a future version of this manual.

V	Code	Warning / Error Text	Description / Workaround
	W001	Could not open include file.	
	W002	No zip codes file defined.	
	W003	No highways file defined.	
	W004	Cannot determine type of element, type cannot be defined before RGNTYPE.	
	W005	Error reading data.	
	W006	Null exit facility name for RGN10 element.	
	W007	Wrong coordinates.	E.g. incorrectly separated coordinates like (2.2,3.2),,(2.2,2.3)
	W008	Element spans more than 10 degrees!	
	W009	Invalid [WPT] section - RGNTYPE is not defined.	
	W010	ELEVATION parameter is depreciated.	
	W011	Invalid [PLT] section - RGNTYPE is not defined.	
	W012	- no longer used	
	W013	TRESIZE smaller than 100.	
	W014	RGNLIMIT should not be smaller than 500.	A small RGNLIMIT derives in the creation of a bigger map, with no gain in speed.
	W015	TRESIZE larger than 5000 - TRESIZE is fixed (i.e. the TRESIZE is automatically changed to 5000).	When the TreSize value in [IMG ID] is too big - the resulting map IMG file renders more and more slowly on a GPS receiver. This value should never be higher than 5000. If the value is larger than 5000, it is internally changed to 5000 and the warning is shown. Note that for a preview map, this warning won't be shown. There is a new key in [IMG ID] to specify that a preview map is created ('Preview=Y').
	W016	- no longer used	
	W017	- no longer used	

V	Code	Warning / Error Text	Description / Workaround
	W018	ID of map should be larger than 0x10000 (65536) or may not work in MapSource.	
	W019	More than ONE background object defined - switching to full manual background creation mode.	This error means that more than one [BACKGROUND] object is defined. Defining more than one background object is not recommended.
	W020	Object has more than 255 nodes - it is allowed only for the PREVIEW map - did you forget to add 'Preview=Y' in [IMG ID]?.	
	W021	Missing [_DRAWORDER] section - polygons may not be visible.	
	W022	DRAWPRIORITY must be between 0 and 31.	
	W023	Wrong index information. Search by city / region / country will NOT work.	Each city which is intended to be used as attribute for POI or road must has its representation as object (RGN20) object – otherwise index for search functionality will not work
	W24	Type of element should not be 0.	
	E001	Could not open file with country name definitions.	
	E002	Invalid name for Country Field in [DEFINITIONS].	
	E003	Could not open file with region name definitions.	
	E004	Invalid name for Region Field in [DEFINITIONS].	
	E005	Invalid name for Region Country Idx in [DEFINITIONS].	
	E006	Could not open file with city name definitions.	
	E007	Invalid name for City Field in [DEFINITIONS].	
	E008	Invalid name for City Region Idx in [DEFINITIONS].	
	E009	Invalid name for ZipCode Field in [DEFINITIONS].	
	E010	Invalid name for Highway Region Idx in [DEFINITIONS].	

V	Code	Warning / Error Text	Description / Workaround
	E011	Invalid name for Highway Field in [DEFINITIONS].	
	E012	Invalid sequence in [COUNTRIES].	
	E013	Invalid sequence in [REGIONS].	
	E014	Invalid sequence in [CITIES].	
	E015	Invalid sequence in [ZIPCODES].	
	E016	Invalid sequence in [HIGHWAYS].	
	E017	Cannot parse coordinates.	e.g. (2.3.4,2.2)
	E018	Type of element for RGN40/RGN80 cannot be higher than 128.	
₪	E019	Street cannot intersect with itself! Split this element!	Only applies to a routable map.
	E020	No more than 8 active layers allowed.	
	E021	Grid definition for layers must be descending (check Level# keys in [IMG ID]).	
	E022	STREETNUMBERSSTART and STREETNUMBERSEND keys are no longer supported - use ROADID instead.	
	E023	Layer detail level too high to cover non-splittable objects from lower layer - decrease detail level (use higher Level#).	This error is similar to error E024, but this error applies to preview maps. The maximum size of any object strictly depends on the bit resolution. For resolution 24, the maximum size ≈ 1.5 metre * 65535. Similarly, for resolution 23, the maximum size ≈ 3 metre * 65535. This means that if the object is too big to fit into the given layer of the map, the bit resolution of this layer needs to be decreased so the layer can accept bigger objects.

V	Code	Warning / Error Text	Description / Workaround
	E024	Top layer detail level too high to cover entire map - decrease detail level of the less detail layer (use higher Level#).	This error needs a little more explanation since it is often a source of confusion. The last layer (the empty one) must always have one 'tre region'. The maximum size of this region is 65535/2 * resolution (grid). If the map covers a large area, the selected grid may be too low to allow the desired 'tre region' to be created. The maximum size of any object strictly depends on the bit resolution. For resolution 24, the maximum size ≈ 1.5 metre * 65535. Similarly, for resolution 23, the maximum size ≈ 3 meter * 65535. This means that if the object is too big to fit into the given layer of the map, the bit resolution of this layer needs to be decreased so the layer can accept bigger objects.
	E025	Zoom definition for layers must be ascending (check Zoom# keys in [IMG ID]).	
	E026	More than 65535 Tre regions were created in a single layer - use bigger TRESIZE and RGNLIMIT or split your map.	
	E027	Timeout limit - compilation interrupted because of the timeout set by administrator	Used only in the Mapcenter special version
	E028		Region and Country information defined by HIGHWAY is not consistent with definition of CITY
	E029	ID of map is not an integer value.	
	E030	Name of the file for preview must be composed always from 8 digits	
	E031	For the preview creation name '00000008.img' is not permitted	
	E032	Layer 0 of the map cannot be empty	Most detailed layer of the map cannot be empty
	E033	Less than 2 layers not allowed	
	E034	Not enough columns for XPM bitmap definition	

V	Code	Warning / Error Text	Description / Workaround
	E035	Wrong XPM bitmap definition	
	E036	ID of map cannot be higher than 268435455 (0x0FFFFFFF)	
	E037	City index beyond the number of defined cities.	
	E038		
₪	R001	Cannot find segment for routing.	
₪	R002	Routing between same points.	
₪	R003	Routable object cannot be filtered - check your [DICTIONARY] section.	
₪	R004	Removing element which can be routable.	
₪	R005	Maximum allowed NODID value is 1048575.	
₪	R006	Creating connections error.	
₪	R007	Node reduction.	
₪	R008	Too short road to be routable - coordinates were aligned to same place	
₪	R010	No data for routing - remove 'ROUTING=Y' from [IMG ID] for non routable maps!	
₪	R011	NODID points cannot be closer than 5.4 meter!	
₪	R012	NODID point defined for non existing point of the road!	
₪	R013	Restriction defined for non existing NODID	
₪	R014	Several DataX key for routable road is not permitted	

12.2 Exits

12.2.1 Valid exit facility types

Mnemonic	Description
0x00	Truck/Lorry Stop / 24-hour Diesel Fuel With Restaurant
0x01	HGV / Diesel Fuel With Large Vehicle Clearance
0x02	Fuel

Mnemonic	Description
0x03	Food / Restaurant
0x04	Lodging / Hotel / Motel
0x05	Auto service / Vehicle Repair and Service
0x06	Auto service / Diesel Engine Service
0x07	Auto service / Commercial Vehicle Wash
0x08	Camp / Campground and RV Service
0x09	Hospital / Medical Facilities
0x0a	Store / Automated Teller Machines
0x0b	Park / Forest, Park, Preserve, or Lake
0x0c	Point Of Interest / Useful Services, Sites, or Attractions
0x0d	Fast Food

12.2.2 Directions

Mnemonic	Description
N	North of
S	South of
E	East of
W	West of
I	Inner Side of
O	Outer Side of
B	Both Sides of
~~EMPTY~~	

12.2.3 Facilities

Facilities can be combined - i.e. facility with Car Wash + Open 24 Hours is 0x48

Mnemonic	Description
0x01	HGV/RV Parking
0x02	Convenience Store
0x04	Diesel Fuel
0x08	Car Wash
0x10	Liquid Propane
0x20	HGV Scales
0x40	Open 24 Hours
0x80	not used

12.3 cGPSmapper object types list

The list below contains the map element types and their associated codes in both hexadecimal and decimal format. This list is distributed with cGPSmapper in two formats: a text file (RGNtype.txt), and an Excel spreadsheet (RGNtype.xls). Both of these files can be found in the cGPSmapper installation directory. The Excel spreadsheet contains a graphical representation of many of the element types.

When a "Y" is present in the marine column (represented with a ⚓), it indicates that the element is only valid when either:

- in the `[IMG ID]` section, there is a definition `Marine=Y`
- in the element definition section (`[POI]` / `[POLYLINE]` / `[POLYGON]`), there is a definition `Marine=Y`

12.3.1 [POI] types

⚓	Code (Hex)	Code (Decimal)	Filter (Dec.)	Find (GPS)	Description
N	0x0100-0x0500	256-1280	1-5		City name (Point, fat, big)
N	0x0600-0x0A00	1536-2560	6-10		City name (Point, big)
N	0x0B00	2816	11		City name (Point, small)
N	0x0C00	3072	12		City name (Point, small)
N	0x0D00	3328	13		City name (Point, small)
N	0x0E00-0x1100	3584-4352	14-17		City name (Point, big)
N	0x1400-0x153F	5120-5439	20-21		Region name (no Point, big)
N	0x1E00-0x1E3F	7680-7743	30		Region name (no Point, middle)
N	0x2000-0x203F	8192-8255	32		Exit
N	0x210F	8463	33		Exit (Service)
N	0x2100-0x213F	8448-8511	33		Exit (with facilities)
N	0x2200-0x223F	8704-8767	34		Exit (Restroom)
N	0x2300-0x233F	8960-9023	35		Exit (Convenience Store)
N	0x2400-0x243F	9216-9279	36		Exit (Weight Station)
N	0x2500-0x253F	9472-9535	37		Exit (Toll Booth)
N	0x2600-0x263F	9728-9791	38		Exit (Information)
N	0x2700-0x273F	9984-10047	39		Exit
N	0x2800-0x283F	10240-10303	40		Region name (no Point, small)
N	0x2A00	10752	42		Dining (Other)
N	0x2A01	10753	42		Dining (American)
N	0x2A02	10754	42		Dining (Asian)
N	0x2A03	10755	42		Dining (Barbecue)
N	0x2A04	10756	42		Dining (Chinese)
N	0x2A05	10757	42		Dining (Deli/Bakery)
N	0x2A06	10758	42		Dining (International)
N	0x2A07	10759	42		Fast Food
N	0x2A08	10760	42		Dining (Italian)
N	0x2A09	10761	42		Dining (Mexican)
N	0x2A0A	10762	42		Dining (Pizza)
N	0x2A0B	10763	42		Dining (Sea Food)
N	0x2A0C	10764	42		Dining (Steak/Grill)
N	0x2A0D	10765	42		Dining (Bagel/Donut)
N	0x2A0E	10766	42		Dining (Cafe/Diner)
N	0x2A0F	10767	42		Dining (French)
N	0x2A10	10768	42		Dining (German)

↗	Code (Hex)	Code (Decimal)	Filter (Dec.)	Find (GPS)	Description
N	0x2A11	10769	42		Dining (British Isles)
N	0x2B00	11008	43		Hotel (Other)
N	0x2B01	11009	43		Hotel/Motel
N	0x2B02	11010	43		Bed & Breakfast inn
N	0x2B03	11011	43		Camping/RV-Park
N	0x2B04	11012	43		Resort
N	0x2C01	11265	44		Amusement Park
N	0x2C02	11266	44		Museum/History
N	0x2C03	11267	44		Library
N	0x2C04	11268	44		Land Mark
N	0x2C05	11269	44		School
N	0x2C06	11270	44		Park
N	0x2C07	11271	44		Zoo
N	0x2C08	11272	44		Sport spark, Stadium (point)
N	0x2C09	11273	44		Fair, Conference (point)
N	0x2C0A	11274	44		Vineyard/Winery (point)
N	0x2C0B	11275	44		Place of Worship
N	0x2C0C	11276	44		Hot Spring
N	0x2D01	11521	45		Theatre
N	0x2D02	11522	45		Bar
N	0x2D03	11523	45		Cinema
N	0x2D04	11524	45		Casino
N	0x2D05	11525	45		Golf
N	0x2D06	11526	45		Ski Centre
N	0x2D07	11527	45		Bowling
N	0x2D08	11528	45		Ice/Sporting
N	0x2D09	11529	45		Swimming
N	0x2D0A	11530	45		Sports (point)
N	0x2D0B	11531	45		Sport Airport
N	0x2E01	11777	46		Department Store
N	0x2E02	11778	46		Grocery
N	0x2E03	11779	46		General Merchandiser
N	0x2E04	11780	46		Shopping Centre
N	0x2E05	11781	46		Pharmacy
N	0x2E06	11782	46		Convenience Store
N	0x2E07	11783	46		Apparel
N	0x2E08	11784	46		House and Garden
N	0x2E09	11785	46		Home Furnishing
N	0x2E0a	11786	46		Special Retail
N	0x2E0b	11787	46		Computer/Software
N	0x2F00	12032	47		Generic Service
N	0x2F01	12033	47		Fuel/Gas

↗	Code (Hex)	Code (Decimal)	Filter (Dec.)	Find (GPS)	Description
N	0x2F02	12034	47		Car Rental
N	0x2F03	12035	47		Car Repair
N	0x2F04	12036	47		Airport
N	0x2F05	12037	47		Post Office
N	0x2F06	12038	47		Bank
N	0x2F07	12039	47		Car Dealer (point)
N	0x2F08	12040	47		Bus Station
N	0x2F09	12041	47		Marina
N	0x2F0A	12042	47		Wrecker Service
N	0x2F0B	12043	47		Parking
N	0x2F0C	12044	47		Restroom
N	0x2F0D	12045	47		Automobile Club
N	0x2F0E	12046	47		Car Wash
N	0x2F0F	12047	47		Garmin Dealer
N	0x2F10	12048	47		Personal Service
N	0x2F11	12049	47		Business Service
N	0x2F12	12050	47		Communication
N	0x2F13	12051	47		Repair Service
N	0x2F14	12052	47		Social Service
N	0x2F15	12053	47		Utility
N	0x2F16	12054	47		Truck/Lorry Stop
N	0x3000	12288	48		Generic Emergency/Government
N	0x3001	12289	48		Police Station
N	0x3002	12290	48		Hospital
N	0x3003	12291	48		Public Office
N	0x3004	12292	48		Justice
N	0x3005	12293	48		Concert hall (point)
N	0x3006	12294	48		Border Station (point)
N	0x4000-0x403F	16384-16447	64		Golf
N	0x4100-0x413F	16640-16703	65		Fish
N	0x4200-0x423F	16896-16959	66		Wreck
N	0x4300-0x433F	17152-17215	67		Marina
N	0x4400-0x443F	17408-17471	68		Gas
N	0x4500-0x453F	17664-17727	69		Restaurant
N	0x4600-0x463F	17920-17983	70		Bar
N	0x4700-0x473F	18176-18239	71		Boat Ramp
N	0x4800-0x483F	18432-18495	72		Camping
N	0x4900-0x493F	18688-18751	73		Park
N	0x4A00-0x4A3F	18944-19007	74		Picnic Area
N	0x4B00-0x4B3F	19200-19263	75		Hospital
N	0x4C00-0x4C3F	19456-19519	76		Information
N	0x4D00-0x4D3F	19712-19775	77		Parking

↗	Code (Hex)	Code (Decimal)	Filter (Dec.)	Find (GPS)	Description
N	0x4E00-0x4E3F	19968-20031	78		Restroom
N	0x4F00-0x4F3F	20224-20287	79		Shower
N	0x5000-0x503F	20480-20543	80		Drinking Water
N	0x5100-0x513F	20736-20799	81		Telephone
N	0x5200-0x523F	20992-21055	82		Scenic Area
N	0x5300-0x533F	21248-21311	83		Skiing
N	0x5400-0x543F	21504-21567	84		Swimming
N	0x5500-0x553F	21760-21823	85		Dam
N	0x5700-0x573F	22272-22335	87		Danger Area
N	0x5800-0x583F	22528-22591	88		Restricted Area
N	0x5900	22784	89		Generic Airport
N	0x5901	22785	89		Large Airport
N	0x5902	22786	89		Medium Airport
N	0x5903	22787	89		Small Airport
N	0x5904	22788	89		Heliport
N	0x5905-0x593F	22789-22847	89		Airport
N	0x5D00-0x5D3F	23808-23871	93		Daymark, Green Square
N	0x5E00-0x5E3F	24064-24127	94		Daymark, Red Triangle
N	0x6200	25088	98		Depth with point one decimal place
N	0x6300	25344	99		Height without point no decimal place
N	0x6400	25600	100		Manmade Feature
N	0x6401	25601	100		Bridge
N	0x6402	25602	100		Building
N	0x6403	25603	100		Cemetery
N	0x6404	25604	100		Church
N	0x6405	25605	100		Civil
N	0x6406	25606	100		Crossing
N	0x6407	25607	100		Dam
N	0x6408	25608	100		Hospital
N	0x6409	25609	100		Levee
N	0x640A	25610	100		Locale
N	0x640B	25611	100		Military
N	0x640C	25612	100		Mine
N	0x640D	25613	100		Oil Field
N	0x640E	25614	100		Park
N	0x640F	25615	100		Post
N	0x6410	25616	100		School
N	0x6411	25617	100		Tower
N	0x6412	25618	100		Trail
N	0x6413	25619	100		Tunnel
N	0x6414	25620	100		Drink water
N	0x6415	25621	100		Ghost Town

↗	Code (Hex)	Code (Decimal)	Filter (Dec.)	Find (GPS)	Description
N	0x6416	25622	100		Subdivision
N	0x6500	25856	101		Water Feature
N	0x6501	25857	101		Arroyo
N	0x6502	25858	101		Sand Bar
N	0x6503	25859	101		Bay
N	0x6504	25860	101		Bend
N	0x6505	25861	101		Canal
N	0x6506	25862	101		Channel
N	0x6507	25863	101		Cove
N	0x6508	25864	101		Falls
N	0x6509	25865	101		Geyser
N	0x650A	25866	101		Glacier
N	0x650B	25867	101		Harbour
N	0x650C	25868	101		Island
N	0x650D	25869	101		Lake
N	0x650E	25870	101		Rapids
N	0x650F	25871	101		Reservoir
N	0x6510	25872	101		Sea
N	0x6511	25873	101		Spring
N	0x6512	25874	101		Stream
N	0x6513	25875	101		Swamp
N	0x6600	26112	102		Land Feature
N	0x6601	26113	102		Arch
N	0x6602	26114	102		Area
N	0x6603	26115	102		Basin
N	0x6604	26116	102		Beach
N	0x6605	26117	102		Bench
N	0x6606	26118	102		Cape
N	0x6607	26119	102		Cliff
N	0x6608	26120	102		Crater
N	0x6609	26121	102		Flat
N	0x660A	26122	102		Forest
N	0x660B	26123	102		Gap
N	0x660C	26124	102		Gut
N	0x660D	26125	102		Isthmus
N	0x660E	26126	102		Lava
N	0x660F	26127	102		Pillar
N	0x6610	26128	102		Plain
N	0x6611	26129	102		Range
N	0x6612	26130	102		Reserve
N	0x6613	26131	102		Ridge
N	0x6614	26132	102		Rock

↗	Code (Hex)	Code (Decimal)	Filter (Dec.)	Find (GPS)	Description
N	0x6615	26133	102		Slope
N	0x6616	26134	102		Summit
N	0x6617	26135	102		Valley
N	0x6618	26136	102		Woods
N	0x1C00	7168	28		Unclassified Obstruction
N	0x1C01	7169	28		Wreck
N	0x1C02	7170	28		Submerged Wreck, dangerous
N	0x1C03	7171	28		Submerged Wreck, non-dangerous
N	0x1C04	7172	28		Wreck, cleared by Wire-drag
N	0x1C05	7173	28		Obstruction, visible at high Water
N	0x1C06	7174	28		Obstruction, awash
N	0x1C07	7175	28		Obstruction, submerged
N	0x1C08	7176	28		Obstruction, cleared by Wire-drag
N	0x1C09	7177	28		Rock, awash
N	0x1C0A	7178	28		Rock, submerged at low Water
N	0x1C0B	7179	28		Sounding
N	0x1D01	7425	29		Tide Prediction
N	0x1B01	6913	27		Fog Horn
N	0x1A01	6657	26		Fog Horn
N	0x1901	6401	25		Fog Horn
N	0x1801	6145	24		Fog Horn
N	0x1701	5889	23		Fog Horn
N	0x1601	5633	22		Fog Horn
N	0x1B02	6914	27		Radio Beacon
N	0x1A02	6658	26		Radio Beacon
N	0x1902	6402	25		Radio Beacon
N	0x1802	6146	24		Radio Beacon
N	0x1702	5890	23		Radio Beacon
N	0x1602	5634	22		Radio Beacon
N	0x1B03	6915	27		Racon
N	0x1A03	6659	26		Racon
N	0x1903	6403	25		Racon
N	0x1803	6147	24		Racon
N	0x1703	5891	23		Racon
N	0x1603	5635	22		Racon
N	0x1B04	6916	27		Daybeacon, red Triangle
N	0x1A04	6660	26		Daybeacon, red Triangle
N	0x1904	6404	25		Daybeacon, red Triangle
N	0x1804	6148	24		Daybeacon, red Triangle
N	0x1704	5892	23		Daybeacon, red Triangle
N	0x1604	5636	22		Daybeacon, red Triangle
N	0x1B05	6917	27		Daybeacon, green Square

↗	Code (Hex)	Code (Decimal)	Filter (Dec.)	Find (GPS)	Description
N	0x1A05	6661	26		Daybeacon, green Square
N	0x1905	6405	25		Daybeacon, green Square
N	0x1805	6149	24		Daybeacon, green Square
N	0x1705	5893	23		Daybeacon, green Square
N	0x1605	5637	22		Daybeacon, green Square
N	0x1B06	6918	27		Daybeacon, white Diamond
N	0x1A06	6662	26		Daybeacon, white Diamond
N	0x1906	6406	25		Daybeacon, white Diamond
N	0x1806	6150	24		Daybeacon, white Diamond
N	0x1706	5894	23		Daybeacon, white Diamond
N	0x1606	5638	22		Daybeacon, white Diamond
N	0x1B07	6919	27		unlit Navaid, white
N	0x1A07	6663	26		unlit Navaid, white
N	0x1907	6407	25		unlit Navaid, white
N	0x1807	6151	24		unlit Navaid, white
N	0x1707	5895	23		unlit Navaid, white
N	0x1607	5639	22		unlit Navaid, white
N	0x1B08	6920	27		unlit Navaid, red
N	0x1A08	6664	26		unlit Navaid, red
N	0x1908	6408	25		unlit Navaid, red
N	0x1808	6152	24		unlit Navaid, red
N	0x1708	5896	23		unlit Navaid, red
N	0x1608	5640	22		unlit Navaid, red
N	0x1B09	6921	27		unlit Navaid, green
N	0x1A09	6665	26		unlit Navaid, green
N	0x1909	6409	25		unlit Navaid, green
N	0x1809	6153	24		unlit Navaid, green
N	0x1709	5897	23		unlit Navaid, green
N	0x1609	5641	22		unlit Navaid, green
N	0x1B0A	6922	27		unlit Navaid, black
N	0x1A0A	6666	26		unlit Navaid, black
N	0x190A	6410	25		unlit Navaid, black
N	0x180A	6154	24		unlit Navaid, black
N	0x170A	5898	23		unlit Navaid, black
N	0x160A	5642	22		unlit Navaid, black
N	0x1B0B	6923	27		unlit Navaid, yellow or amber
N	0x1A0B	6667	26		unlit Navaid, yellow or amber
N	0x190B	6411	25		unlit Navaid, yellow or amber
N	0x180B	6155	24		unlit Navaid, yellow or amber
N	0x170B	5899	23		unlit Navaid, yellow or amber
N	0x160B	5643	22		unlit Navaid, yellow or amber
N	0x1B0C	6924	27		unlit Navaid, orange

↗	Code (Hex)	Code (Decimal)	Filter (Dec.)	Find (GPS)	Description
N	0x1A0C	6668	26		unlit Navaid, orange
N	0x190C	6412	25		unlit Navaid, orange
N	0x180C	6156	24		unlit Navaid, orange
N	0x170C	5900	23		unlit Navaid, orange
N	0x160C	5644	22		unlit Navaid, orange
N	0x1B0D	6925	27		unlit Navaid, multi coloured
N	0x1A0D	6669	26		unlit Navaid, multi coloured
N	0x190D	6413	25		unlit Navaid, multi coloured
N	0x180D	6157	24		unlit Navaid, multi coloured
N	0x170D	5901	23		unlit Navaid, multi coloured
N	0x160D	5645	22		unlit Navaid, multi coloured
N	0x1B0E	6926	27		Navaid, unknown
N	0x1A0E	6670	26		Navaid, unknown
N	0x190E	6414	25		Navaid, unknown
N	0x180E	6158	24		Navaid, unknown
N	0x170E	5902	23		Navaid, unknown
N	0x160E	5646	22		Navaid, unknown
N	0x1B0F	6927	27		lighted Navaid, white
N	0x1A0F	6671	26		lighted Navaid, white
N	0x190F	6415	25		lighted Navaid, white
N	0x180F	6159	24		lighted Navaid, white
N	0x170F	5903	23		lighted Navaid, white
N	0x160F	5647	22		lighted Navaid, white
N	0x1B10	6928	27		lighted Navaid, red
N	0x1A10	6672	26		lighted Navaid, red
N	0x1910	6416	25		lighted Navaid, red
N	0x1810	6160	24		lighted Navaid, red
N	0x1710	5904	23		lighted Navaid, red
N	0x1610	5648	22		lighted Navaid, red
N	0x1B11	6929	27		lighted Navaid, green
N	0x1A11	6673	26		lighted Navaid, green
N	0x1911	6417	25		lighted Navaid, green
N	0x1811	6161	24		lighted Navaid, green
N	0x1711	5905	23		lighted Navaid, green
N	0x1611	5649	22		lighted Navaid, green
N	0x1B12	6930	27		lighted Navaid, yellow or amber
N	0x1A12	6674	26		lighted Navaid, yellow or amber
N	0x1912	6418	25		lighted Navaid, yellow or amber
N	0x1812	6162	24		lighted Navaid, yellow or amber
N	0x1712	5906	23		lighted Navaid, yellow or amber
N	0x1612	5650	22		lighted Navaid, yellow or amber
N	0x1B13	6931	27		lighted Navaid, orange

↗	Code (Hex)	Code (Decimal)	Filter (Dec.)	Find (GPS)	Description
N	0x1A13	6675	26		lighted Navaid, orange
N	0x1913	6419	25		lighted Navaid, orange
N	0x1813	6163	24		lighted Navaid, orange
N	0x1713	5907	23		lighted Navaid, orange
N	0x1613	5651	22		lighted Navaid, orange
N	0x1B14	6932	27		lighted Navaid, violet
N	0x1A14	6676	26		lighted Navaid, violet
N	0x1914	6420	25		lighted Navaid, violet
N	0x1814	6164	24		lighted Navaid, violet
N	0x1714	5908	23		lighted Navaid, violet
N	0x1614	5652	22		lighted Navaid, violet
N	0x1B15	6933	27		lighted Navaid, blue
N	0x1A15	6677	26		lighted Navaid, blue
N	0x1915	6421	25		lighted Navaid, blue
N	0x1815	6165	24		lighted Navaid, blue
N	0x1715	5909	23		lighted Navaid, blue
N	0x1615	5653	22		lighted Navaid, blue
N	0x1B16	6934	27		lighted Navaid, multi coloured
N	0x1A16	6678	26		lighted Navaid, multi coloured
N	0x1916	6422	25		lighted Navaid, multi coloured
N	0x1816	6166	24		lighted Navaid, multi coloured
N	0x1716	5910	23		lighted Navaid, multi coloured
N	0x1616	5654	22		lighted Navaid, multi coloured
Y	0x0100	256	1	N	Light
Y	0x0102	258	1	N	Light with north topmark
Y	0x0103	259	1	N	Light with south topmark
Y	0x0104	260	1	N	Light with east topmark
Y	0x0105	261	1	N	Light with west topmark
Y	0x0106	262	1	N	Isolated danger light
Y	0x0107	263	1	N	Port hand light
Y	0x0108	264	1	N	Starboard hand light
Y	0x0109	265	1	N	Special purpose light
Y	0x010a	266	1	N	Safe water light
Y	0x0200	512	2	N	Buoy
Y	0x0201	513	2	N	Buoy
Y	0x0202	514	2	N	Buoy with north topmark
Y	0x0203	515	2	N	Buoy with south topmark
Y	0x0204	516	2	N	Buoy with east topmark
Y	0x0205	517	2	N	Buoy with west topmark
Y	0x0206	518	2	N	Beacon
Y	0x0207	519	2	N	Spar buoy
Y	0x0208	520	2	N	Isolated danger buoy

↗	Code (Hex)	Code (Decimal)	Filter (Dec.)	Find (GPS)	Description
Y	0x0209	521	2	N	Port hand buoy
Y	0x020a	522	2	N	Starboard hand buoy
Y	0x020b	523	2	N	Special purpose buoy
Y	0x020c	524	2	N	Safe water buoy
Y	0x020d	525	2	N	Platform buoy
Y	0x020e	526	2	N	Beacon with north topmark
Y	0x020f	527	2	N	Beacon with south north topmark
Y	0x0210	528	2	N	Beacon with east topmark
Y	0x0211	529	2	N	Beacon with west topmark
Y	0x0212	530	2	N	Isolated danger beacon
Y	0x0213	531	2	N	Port hand beacon
Y	0x0214	532	2	N	Starboard hand beacon
Y	0x0215	533	2	N	Special purpose beacon
Y	0x0216	534	2	N	Mooring buoy
Y	0x0217	535	2	N	Fixed point
Y	0x0218	536	2	N	Pole
Y	0x0300	768	3	N	Depth point
Y	0x0301	769	3	N	Depth point invisible
Y	0x0302	770	3	N	Depth point underscore
Y	0x0303	771	3	N	Spot height
Y	0x0304	772	3	N	Building
Y	0x0305	773	3	N	Chimney
Y	0x0306	774	3	N	Church
Y	0x0307	775	3	N	Tanks
Y	0x0308	776	3	N	Tower
Y	0x0309	777	3	N	Rock
Y	0x030a	778	3	N	Triangulation point
Y	0x030b	779	3	N	Radio mast
Y	0x0400	1024	4	Y	Isolated danger
Y	0x0401	1025	4	Y	Obstruction
Y	0x0402	1026	4	Y	Wreck
Y	0x0403	1027	4	Y	Exposed wreck
Y	0x0404	1028	4	Y	Well
Y	0x0405	1029	4	Y	Foul
Y	0x0406	1030	4	Y	Explosive
Y	0x0407	1031	4	Y	Fish haven
Y	0x0408	1032	4	Y	Obstruction that covers
Y	0x0409	1033	4	Y	Marine farm
Y	0x040a	1034	4	Y	Dangerous rock
Y	0x040b	1035	4	Y	No bottom found
Y	0x040c	1036	4	Y	Exposed rock
Y	0x040d	1037	4	Y	Dangerous rock

↗	Code (Hex)	Code (Decimal)	Filter (Dec.)	Find (GPS)	Description
Y	0x040e	1038	4	Y	Underwater rock (non-dangerous rock)
Y	0x040f	1039	4	Y	Shoal
Y	0x0500	1280	5	N	Label point
Y	0x0600	1536	6	N	Centred label
Y	0x0700	1792	7	N	Miscellaneous point
Y	0x0701	1793	7	Y	Recommended anchorage
Y	0x0702	1794	7	N	Pilot boarding place
Y	0x0703	1795	7	N	Yacht harbour
Y	0x0704	1796	7	N	Pile
Y	0x0705	1797	7	Y	Anchoring prohibited
Y	0x0706	1798	7	Y	Fishing prohibited
Y	0x0707	1799	7	Y	Precautionary area
Y	0x0708	1800	7	N	Radio report point
Y	0x0709	1801	7	N	Anchorage berths
Y	0x070a	1802	7	N	Rescue station
Y	0x070b	1803	7	N	Fishing harbour
Y	0x070c	1804	7	N	Airport
Y	0x0801	2049	8	N	Information
Y	0x0901	2305	9	N	Bottom conditions
Y	0x0902	2306	9	N	Fishing information
Y	0x0903	2307	9	N	Facility

12.3.2 [POLYLINE] types

↗	Code (Hex)	Code (Decimal)	Description
N	0x01	1	Major Highway-thick
N	0x02	2	Principal Highway-thick
N	0x03	3	Principal Highway-medium
N	0x04	4	Arterial Road–medium
N	0x05	5	Arterial Road-thick
N	0x06	6	Road-thin
N	0x07	7	Alley-thick
N	0x08	8	Ramp
N	0x09	9	Ramp
N	0x0a	10	Unpaved Road-thin
N	0x0b	11	Major Highway Connector-thick
N	0x0c	12	Roundabout
N	0x14	20	Railroad
N	0x15	21	Shoreline
N	0x16	22	Track/Trail
N	0x18	24	Stream-thin
N	0x19	25	Time-Zone

↗	Code (Hex)	Code (Decimal)	Description
N	0x1a	26	Ferry
N	0x1b	27	Ferry
N	0x1c	28	Political Boundary
N	0x1d	29	County Boundary
N	0x1e	30	International Boundary
N	0x1f	31	River
N	0x20	32	Land Contour (thin)
N	0x21	33	Land Contour (medium)
N	0x22	34	Land Contour (thick)
N	0x23	35	Depth Contour (thin)
N	0x24	36	Depth Contour (medium)
N	0x25	37	Depth Contour (thick)
N	0x26	38	Intermittent River
N	0x27	39	Airport Runway
N	0x28	40	Pipeline
N	0x29	41	Power line
N	0x2a	42	Marine Boundary (no line)
N	0x2b	43	Marine Hazard (no line)
Y	0x0100	256	Miscellaneous line
Y	0x0101	257	Line
Y	0x0102	258	Cartographic line
Y	0x0103	259	Road
Y	0x0104	260	Clearing line
Y	0x0105	261	Contour line
Y	0x0106	262	Overhead cable
Y	0x0107	263	Bridge
Y	0x0108	264	Recommended route
Y	0x0109	265	Chart border
Y	0x0300	768	Depth contour
Y	0x0301	769	Depth contour value
Y	0x0307	775	Intertidal zone border
Y	0x0400	1024	Obstruction line
Y	0x0401	1025	Submarine cable
Y	0x0402	1026	Submarine pipeline
Y	0x0403	1027	Pile barrier
Y	0x0404	1028	Fishing stakes
Y	0x0405	1029	Supply pipeline area
Y	0x0406	1030	Submarine cable area
Y	0x0407	1031	Dumping ground
Y	0x0408	1032	Explosive dumping ground
Y	0x0409	1033	Danger line
Y	0x040a	1034	Overhead cable

↗	Code (Hex)	Code (Decimal)	Description
Y	0x040b	1035	Submerged construction
Y	0x040c	1036	Pier/jetty
Y	0x0500	1280	Restriction
Y	0x0501	1281	Anchoring prohibited
Y	0x0502	1282	Fishing prohibited
Y	0x0503	1283	Prohibited area
Y	0x0504	1284	Military practice area
Y	0x0505	1285	Anchoring and fishing prohibited
Y	0x0506	1286	Limit of nature reservation
Y	0x0507	1287	Restricted area
Y	0x0508	1288	Minefield
Y	0x0600	1536	Miscellaneous line
Y	0x0601	1537	Cartographic line
Y	0x0602	1538	Traffic separation line
Y	0x0603	1539	International maritime boundary
Y	0x0604	1540	Straight territorial sea baseline
Y	0x0605	1541	Seaward limit of territorial sea
Y	0x0606	1542	Anchorage area
Y	0x0607	1543	Quarantine anchorage area
Y	0x0608	1544	Fishery zone
Y	0x0609	1545	Swept area
Y	0x060a	1546	Traffic separation zone
Y	0x060b	1547	Limit of exclusive economic zone
Y	0x060c	1548	Established direction of traffic flow
Y	0x060d	1549	Recommended direction of traffic flow
Y	0x060e	1550	Harbour limit
Y	0x060f	1551	Inadequately surveyed area
Y	0x0610	1552	Inshore traffic zone
Y	0x0611	1553	Limit of traffic lane
Y	0x0701	1793	River channel
Y	0x0702	1794	Submerged object
	...		
Y	0x0706	1798	Chart boundary

12.3.3 [POLYGON] types

↗	Code (Hex)	Code (Decimal)	Description
N	0x01	1	City
N	0x02	2	City
N	0x03	3	City
N	0x04	4	Military
N	0x05	5	Car Park (Parking Lot)

♂	Code (Hex)	Code (Decimal)	Description
N	0x06	6	Parking Garage
N	0x07	7	Airport
N	0x08	8	Shopping Centre
N	0x09	9	Marina
N	0x0a	10	University
N	0x0b	11	Hospital
N	0x0c	12	Industrial
N	0x0d	13	Reservation
N	0x0e	14	Airport Runway
N	0x13	19	Man made area
N	0x14	20	National park
N	0x15	21	National park
N	0x16	22	National park
N	0x17	23	City Park
N	0x18	24	Golf
N	0x19	25	Sport
N	0x1a	26	Cemetery
N	0x1e	30	State Park
N	0x1f	31	State Park
N	0x28	40	Ocean
N	0x3b	59	Blue-Unknown
N	0x32	50	Sea
N	0x3b	59	Blue-Unknown
N	0x3c	60	Lake
N	0x3d	61	Lake
N	0x3e	62	Lake
N	0x3f	63	Lake
N	0x40	64	Lake
N	0x41	65	Lake
N	0x42	66	Lake
N	0x43	67	Lake
N	0x44	68	Lake
N	0x45	69	Blue-Unknown
N	0x46	70	River
N	0x47	71	River
N	0x48	72	River
N	0x49	73	River
N	0x4b	75	Background
N	0x4c	76	Intermittent River/Lake
N	0x4d	77	Glacier
N	0x4e	78	Orchard or plantation
N	0x4f	79	Scrub

↗	Code (Hex)	Code (Decimal)	Description
N	0x50	80	Woods
N	0x51	81	Wetland
N	0x52	82	Tundra
N	0x53	83	Flats
Y	0x0100	256	Land - white
Y	0x0101	257	Land - non-urban
Y	0x0102	258	Land - urban
Y	0x0103	259	Chart exclusion area
Y	0x0104	260	Chart background
Y	0x0105	261	Bridge
Y	0x0300	768	Depth area - white 1
Y	0x0301	769	Intertidal zone
Y	0x0302	770	Depth area - blue 1
Y	0x0303	771	Depth area - blue 2
Y	0x0304	772	Depth area - blue 3
Y	0x0305	773	Depth area - blue 4
Y	0x0306	774	Depth area - blue 5
Y	0x0307	775	Depth area - white
Y	0x0400	1024	Obstruction (invisible)
Y	0x0401	1025	Submarine cable (invisible)
Y	0x0402	1026	Submarine pipeline (invisible)
Y	0x0403	1027	Pile barrier (invisible)
Y	0x0404	1028	Fishing stakes (invisible)
Y	0x0405	1029	Supply pipeline area/line (invisible)
Y	0x0406	1030	Submarine cable area/line (invisible)
Y	0x0407	1031	Dumping ground (invisible)
Y	0x0408	1032	Explosive dumping ground (invisible)
Y	0x0409	1033	Danger line (invisible)
Y	0x040a	1034	Overhead cable (invisible)
Y	0x040b	1035	Submerged construction (invisible)
Y	0x040c	1036	Pier/jetty (invisible)
Y	0x0500	1280	Restriction area/line (invisible)
Y	0x0501	1281	Anchoring prohibited (invisible)
Y	0x0502	1282	Fishing prohibited (invisible)
Y	0x0503	1283	Prohibited area (invisible)
Y	0x0504	1284	Military practice area (invisible)
Y	0x0505	1285	Anchoring and fishing prohibited (invisible)
Y	0x0506	1286	Limit of nature reservation (invisible)
Y	0x0507	1287	Restricted area (invisible)
Y	0x0508	1288	Minefield (invisible)
Y	0x0600	1536	Miscellaneous area
Y	0x0601	1537	Cartographic area

♂	Code (Hex)	Code (Decimal)	Description
Y	0x0602	1538	Traffic separation area
Y	0x0603	1539	International maritime boundary
Y	0x0604	1540	Straight territorial sea baseline
Y	0x0605	1541	Seaward limit of territorial sea
Y	0x0606	1542	Anchorage area
Y	0x0607	1543	Quarantine anchorage area
Y	0x0608	1544	Fishery zone
Y	0x0609	1545	Swept area
Y	0x060a	1546	Traffic separation zone
Y	0x060b	1547	Limit of exclusive economic zone
Y	0x060c	1548	Established direction of traffic flow
Y	0x0701	1793	Fishing area
Y	0x0702	1794	Restricted area
Y	0x0703	1795	Anchorage area
Y	0x0704	1796	Fishing Hot Spots chart

12.3.4 Custom types name substitution

You may create up to 4 default names in different languages to be used if the object does not have a label. For example:

```
[_line]
Type=0x01
String1=0x01,Route          ; French
String2=0x02,Landstraße         ; German
String3=0x04,Highway            ; English
String4=0x08,Carretera          ; Spanish
LineWidth=5
BorderWidth=1
xpm="0 0 4 0"               ; Define both day and night
colors (4)
"1 c #20c818"               ; Daytime interior color
"2 c #309838"               ; Daytime border color
"3 c #20c818"               ; Nighttime interior color
"4 c #086808"               ; Nighttime border color
[end]
```

Code	Language		Code	Language
0x00	Unspecified		0x12	Czech
0x01	French		0x13	Croatian
0x02	German		0x14	Hungarian
0x03	Dutch		0x15	Polish
0x04	English		0x16	Turkish
0x05	Italian		0x17	Greek

0x06	Finnish		0x18	Slovenian
0x07	Swedish		0x19	Russian
0x08	Spanish		0x1a	Estonian
0x09	Basque		0x1b	Latvian
0x0a	Catalan		0x1c	Romanian
0x0b	Galican		0x1d	Albanian
0x0c	Welsh		0x1e	Bosnian
0x0d	Gaelic		0x1f	Lithuanian
0x0e	Danish		0x20	Serbian
0x0f	Norwegian		0x21	Macedonian
0x10	Portuguese		0x22	Bulgarian
0x11	Slovak			

12.3.5 How do I create XPM definitions?

If you want to create any other than the simplest shapes for your POIs, you will want to use graphics tools to manage your source bitmaps and ultimate XPM definition. Here's a description of one approach using Photoshop Elements, IconXP and Microsoft Word; this is certainly not the only way.

Photoshop steps:

- Create the original full-color image. You may find it easier to edit the image at a multiple of its target size. For example, 96x96 is a good size, as it scales well to 24x24, 16x16, 12x12 and 8x8 nicely. Or, you can edit at the target dimensions.
- Create your transparent areas as desired.
- Resize as needed to your target dimensions.
- Save in PNG-24 format with transparency.

IconXP steps:

- Go to http://www.aha-soft.com/iconxp/index.htm to download a trial version of IconXP. (The registered version is $20US)
- Open your .PNG file from Photoshop.
- Export As .XPM

Microsoft Word steps:

- Open the .xpm file.
- Look for any instances of color definitions using 'black' or 'white'; replace them with #000000 or #FFFFFF. cGPSmapper does not support these literals.
- Copy the definition into your source file, starting with the quotation mark before the first line of the declaration, all the way to the closing brace.

Notepad steps:

- Add the necessary header, type, strings and [end] statement.

12.4 cGPSmapper versions

The table below lists the various versions of cGPSmapper and illustrates the main differences between each of the versions. For more information, including the latest prices, visit http://www.cgpsmapper.com/.

	Version	
ϕ	Freeware	• No city or POI indexing • No additional city information • No additional POI information • No map copyright • Maps created with this version should not be sold
σ	Shareware	• Direct support for ESRI shape format • City and POI indexing is limited to 100 cities and POIs in standard maps. Indexing means that cities and POIs may be searched using the GPS receiver's "Find by name" function (subject to the receiver limitations). • City and POI indexing is limited to 65,500 cities and POIs in so-called POI maps, i.e. maps containing only cities and POIs, with no dimensional objects (such as roads or forests) (maps created with '-i' switch) • Additional POI information: country, region, city, and description (displayed in the details window), but no phone number and full address • Additional city information: country and region • The purchased copy is registered permanently to the purchaser's name and e-mail address (this information is displayed by the receiver in the map copyright section) • Maps created with this version should not be sold
π	Pro	• Building numbering • Additional city, region and country information for roads and POI • Search by address - street name, house number and optionally zip code and city • Search for intersection • City and POI indexes are not limited, creation of global indexing (multi IMG mapsets) • Full POI address and additional descriptions • 'lock on road' feature • User defined copyright text • Limited support

	Version	
₪	Routable Personal Edition	This version has full scope of functionality – including creation of routable maps - with the following exceptions • Hardcoded copyright string 'name surname email@email.com, cGPSmapper personal edition' • No support for creating routable maps! Only basic support regarding the input data format • No permission for commercial use
₪	Routable	Fully routable maps - find fastest or shortest route, support for all kinds of restrictions and time limited restrictions

12.5 cGPSmapper files

♣ This section will be further documented in a future version of this manual.
The table below lists the contents of the main files that are distributed with the compiler.

File	Contents
`cGPSmapper-Help .txt`	How to obtain further details to use the compiler.
`cgpsmapper.exe`	cGPSmapper compiler binary executable.
`Datum_List.txt`	Full list of supported datums to be used in the `Datum` element. Refer to section 4.2.1 (on page 9) for details.
`Readme.first`	Description of the sample files provided and how to obtain further details to use the compiler.
`Readme0080.txt`	Release notes with details on the improvements made to the compiler.
`RGNtype.txt`	cGPSmapper element types list in plain text format. Refer to section 12.3 (on page 101) for details.
`RGNtype.xls`	cGPSmapper element types list in Excel format. Contains the graphical representation of many of the element types. Refer to section 12.3 (on page 101) for details.
`Strings.txt`	Character coding documentation. Refer to section ♣ (on page ♣) for details.
`Test_Map`	Directory containing a sample map.
`Licence.txt`	Terms of use of the free version of cGPSmapper

13 Index and Tables

13.1 Table of Figures

13.2 Version Control Log

Ver#	Date	Edited by	Section	Changes
1.0	2005-04-01	M. Zalba	-	Initial Release
1.1	2005-04-04	H.Scheffler	2.4	Removed some author names as requested
1.2	2005-05-23	H.Scheffler	-	PDF with higher resolution images
2.0	2005-07-08	M. Zalba	-	Added marine documentation and updated ESRI documentation.
			4.2	PFM syntax Description Added [CHART INFO] section to the end of the Declarations section.
			4.2.1	Header "Marine" element added. "DrawPriority" element added.
			4.2.2.4	Chart Info New section.
			4.2.4.1 4.2.4.2 4.2.4.3	"SubType" element added to Points of Interest, Polygons and Polylines.
			4.2.4.6	*Shapes* Section updated.
			4.3	Marine Charts New section.
			8.1	*cGPSmapper compilation Errors and Warnings* Changed W014 Added: R010, R011, R012
			8.3	*cGPSmapper object types list* Note about the marine objects added.
			8.3.1 8.3.2 8.3.3	Marine objects and their description added. "Find" (GPS) feature added.
			8.4	cGPSmapper versions Marine version added.
2.1	2006-10-10	G.Rikker	5	Custom TYP file

Ver#	Date	Edited by	Section	Changes
4.0	2006-12-01	S.Kozicki	-	Locking with cGPSmapper Corrections
				Codepages
4.4.4	2008			Added [DEFINITIONS] section description
	2009			Corrections

13.3 Index