Getting Started
with
Global Mapper 8
and
cGPSMapper

Greg Riker
October 2006
GRiker98121@yahoo.com

Greetings Mapmakers.

This document will walk you through several tutorials using Global Mapper 8 and cGPSMapper to create custom maps for Garmin GPS receivers. The assumption is that you are a motivated computer user of moderate skill with an interest in making your own digital maps. If so, you've come to the right place. If you get stuck, there are many online resources listed at the end of this document where you can find helpful users of both programs.

Terminology

Some terms and phrases are used very frequently, so let's introduce some shorthand to refer to them:

- **GM8** – Global Mapper 8, software used to create source files for maps.
- **GPSr** – a generic term referring to a GPS receiver. Within this document, GPSr refers specifically to Garmin GPS receivers.
- **.MP** – Polish format source files exported by GM8, imported by cGPSMapper.
- **.IMG** – the compiled file format output by cGPSMapper, which is downloadable into a Garmin GPSr.
- **GME** – GPSMapEdit, software used with .MP source files to define routing information in your maps.

The Tools

Global Mapper 8

GM8 is a software tool from Global Mapper Software (http://GlobalMapper.com) enabling creation of digital maps. Think of it as a word processor for geographic data. With a digital representation of your data, you can create printed maps or digital map files that can be downloaded into Garmin GPSr's. Version 8.1 was used in the preparation of these tutorials.

cGPSMapper

cGPSMapper is a software tool from cGPSMapper (http://cGPSMapper.com) that compiles .MP format source code exported by GM8 into a .IMG file that can be downloaded and displayed in a Garmin GPS receiver. The free version of cGPSMapper supports all features described in this document except routing. There is a 30-day trial version available at the cGPSMapper site allowing you to experiment with routing. Version 0.90 was used during the preparation of these tutorials.

SendMap20

SendMap20 downloads .IMG files created by cGPSMapper into Garmin GPSr's. A free version is available at http://cgpsmapper.com/buy.htm. There is also a Pro version available with an extended feature set. Version 3.5 was used during the preparation of these tutorials.

GPSMapEdit

GME is a software tool from Geopainting.com (http://Geopainting.com). Many of its functions overlap with those of GM8. In this sequence of tutorials, GME is used to add routing data to your maps. Version 1.0.32.0 was used during the preparation of these tutorials.

Text Editor

A text editor can be as simple as Notepad, but a *structured editor* is a much more powerful tool for a mapmaker. While you're getting started, Notepad will be fine, but when you're ready to step up to something more capable I recommend EditPad Pro, available from http://editpadpro.com.

Setting up your computer

Mapmaking is a complex process requiring specialized software tools and a powerful computer. For these tutorials, you are assumed to be operating in the Windows XP environment with at least 512MB of RAM and plenty of hard disk storage. The tutorials will generate about 200MB of data. Larger projects can easily generate several gigabytes of data.

Installing the tools

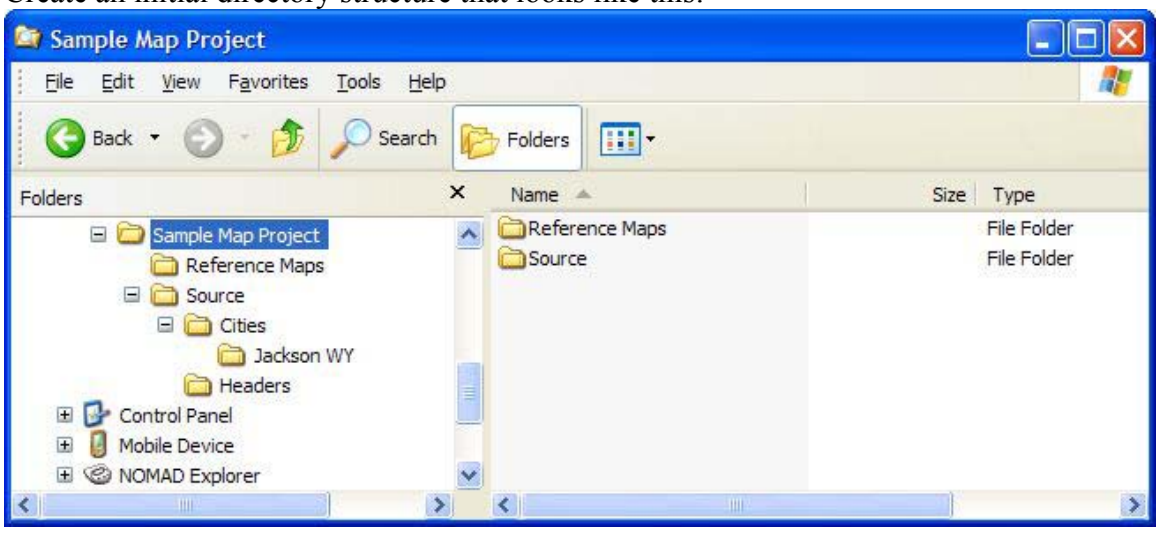
If you haven't already done so, install Global Mapper 8 and cGPSMapper.

Hard disk storage

For this introductory tour, you will need at least 200MB of free hard disk space.

Directory structure

For your map data, you may wish to consider using a separate disk storage device. Create an initial directory structure that looks like this:



Broadband connection

A broadband connection to the Internet is desirable for downloading satellite imagery.

GPS requirements

To display your compiled data, you will need a Garmin GPSr with the ability to display downloaded maps. Additionally, if you want to work with custom types, giving you the ability to control how your features look in the GPSr, you will need a relatively recent model supporting this ability.

Planning Your Project

You may have already experimented with making some digital maps. For this tutorial, we'll assume that you're starting from scratch. You're probably anxious to dive in and

starting doing something, but a little planning up front will make the entire process more rewarding.

The scope of your project will determine what data you need to collect. For our introductory tutorial we will create a map with the three principal data types: Points, Lines and Areas. We will download some aerial imagery from online sources to use as a digitizing reference, and we'll generate some topographic contour lines.

Data sources

All maps are compilations of data from multiple sources, and yours will be no exception. What makes maps unique is the individual perspective that each mapmaker brings to their craft. Your design decisions to emphasize certain features will make your map unique.

> ***A note about copyrights:*** *Copyright is a serious matter. Nothing in this guide is intended to violate anyone's copyrighted material. Understand that there is a clear distinction between facts (which cannot be copyrighted) and the representation of facts (which may be copyrightable). Looking at a copyrighted map to understand the fact that a city or street is located at certain geographic coordinates is fair use of that fact. Photocopying the map and selling it as your own work is not fair use. Taking someone else's compiled map data and representing it as your own work is a crime. Please don't violate these common-sense principles.*

Conventions used in this document

Convention	Meaning
Bold italics	A program command
Pipe (***File*** | ***Open***)	A pipe represents a level in the menu hierarchy. For example, ***File*** | ***Open*** means to click on the File menu, then select the Open command.
SMALL CAPS	Small caps are used for file names (JACKSON.MP) and extensions (.GMW), and Overlays (GENERATED CONTOURS)

Tutorial 1a – Using the Digitizer

Goals for this section:

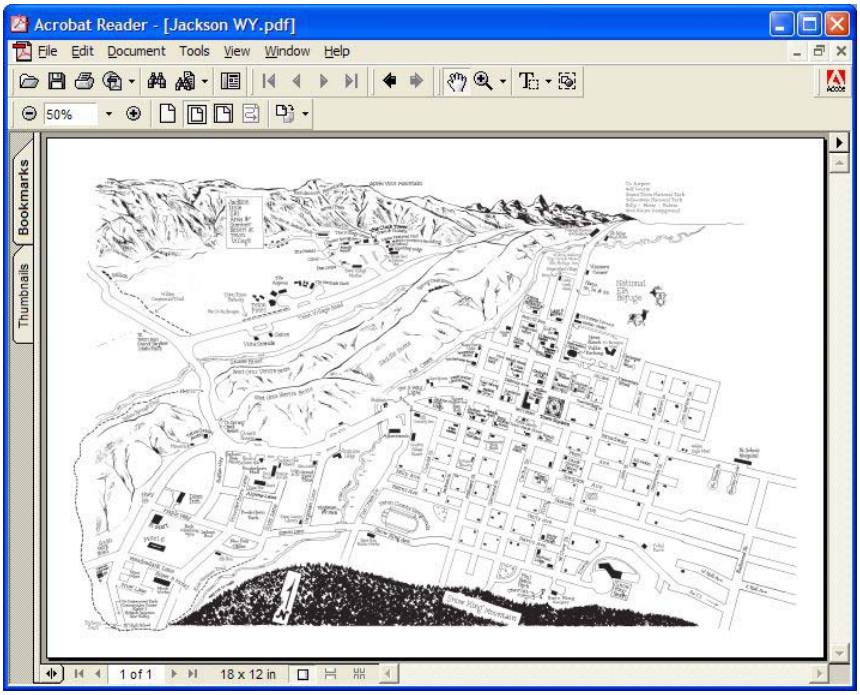
- Download and save aerial imagery from TerraServer.com
- Digitize Lines, Points and Areas
- Save your GM8 workspace
- Export a .MP file for compilation with cGPSMapper
- Download the compiled .IMG file to your GPSr with SendMap20

Let's get started.

GM8 has the ability to load and display overlays concurrently. This means that you may have a reference map loaded in a *raster overlay*, from which you can trace or digitize features in a *vector overlay*. We'll begin the tutorial sequence working with both types of overlays.

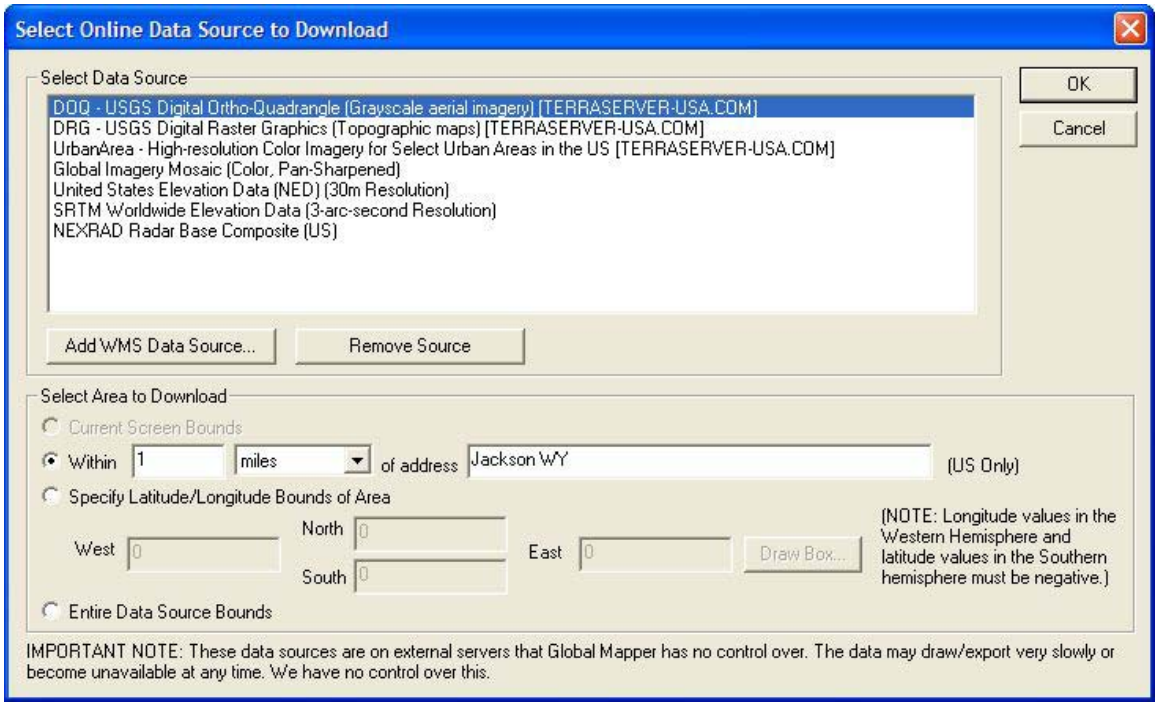
Let's see what data we can find online for Jackson, Wyoming.

I performed a Google search for 'Jackson WY map' and found this useful hand-drawn map in PDF format at
http://www.jacksonholewy.net/images/content/maps/map_jackson_toon.pdf



This is a typical tourist map showing local areas of interest. It's clearly not to scale, and it appears to have been drawn with perspective to emphasize the nearby mountains. But it has lots of local detail that we can use. Open this PDF file in your browser.

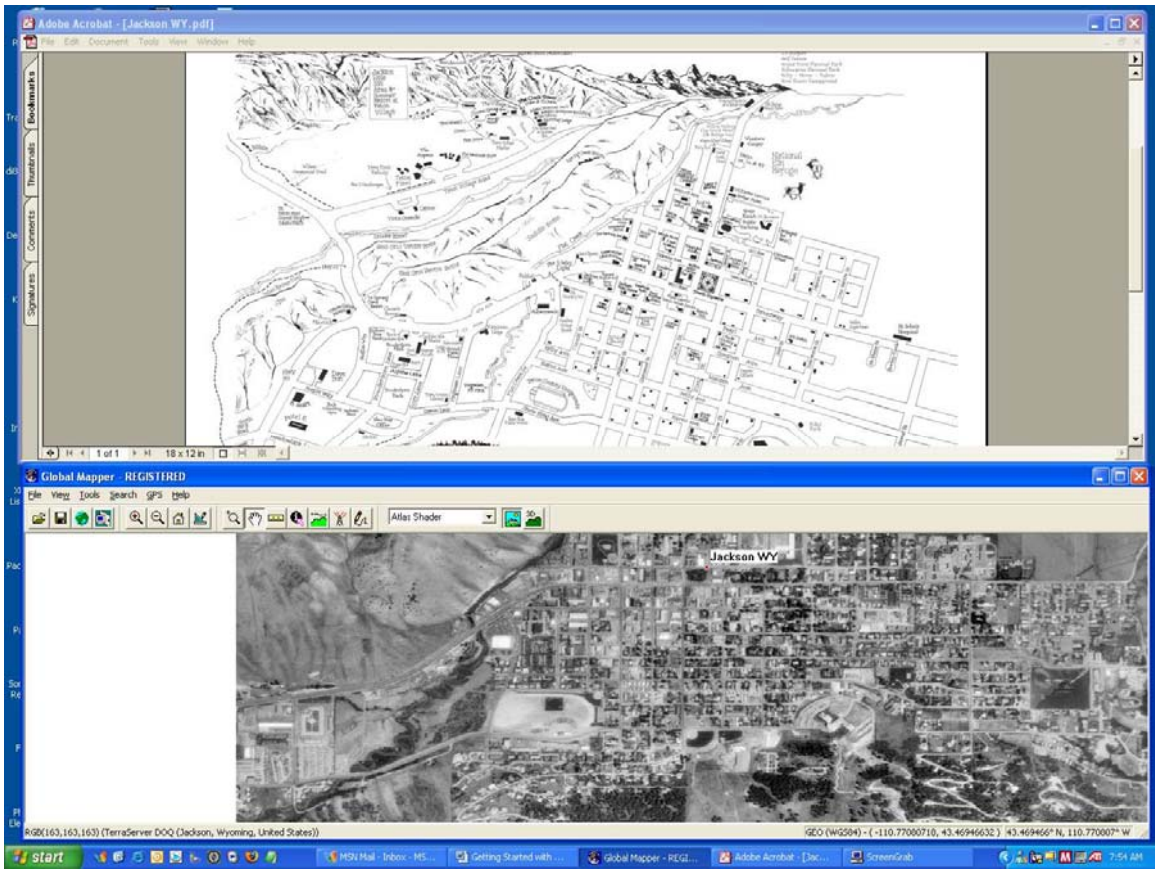
GM8 offers a direct connection to online imagery. Click ***File | Download Online Imagery/Topo Maps***. Select DOQ TerraServer as the source, and in Select Area to Download, specify Within 1 mile of address Jackson WY.



GM8 will download your requested aerial imagery, centered on Jackson WY.

While we're waiting for the imagery to download, note that certain areas of the US have higher resolution color imagery available, and that data from NASA's Shuttle Radar Topography Mission is available for the entire world. GM8 makes it very easy to locate and download useful reference imagery for your mapping projects. Google Earth is another very useful resource.

Let's digitize a few features. The best working arrangement is to be able to see both the PDF tourist map of Jackson and the Global Mapper window at the same time. If you have two displays on your system, use them! If not, I suggest arranging the two windows stacked above and below, since the maps we're working with are oriented similarly in landscape mode.

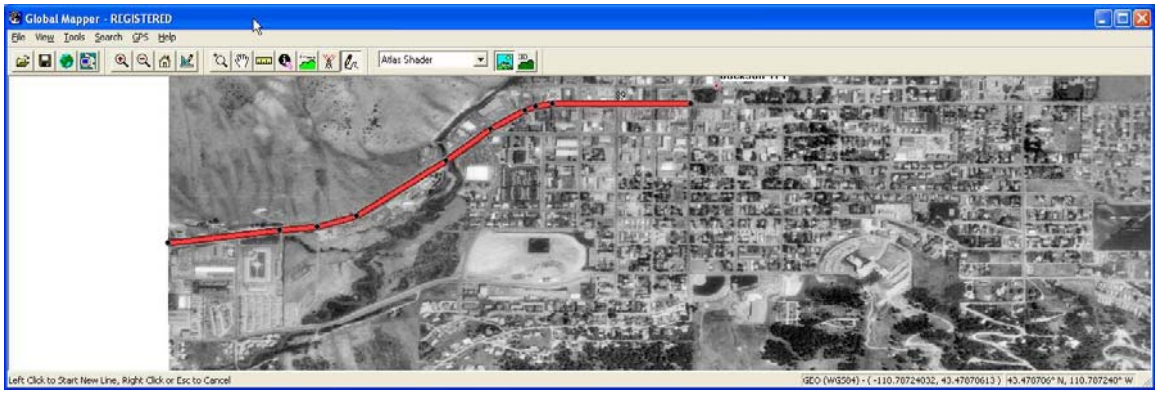


First, we need to establish a reference point between the two maps. We'll find highway 89 and trace its route through town. In the aerial image, there is a large clear area shown just left of center, SW of the 'Jackson WY' dot, which appears to be the Teton County Fairgrounds. In GM8, press ***Alt+Z*** to switch to Zoom Mode, then click and drag over an area including Highway 89, the fairgrounds, and the Town Square, next to the 'Jackson WY' dot.

Referencing the PDF map, we can see that Highway 89 enters from the SW, turns east into town, then exits north just before the Town Square. We'll trace this as our first road. Press ***Alt+D*** to switch to Edit Mode, then right-click and select ***Create New Line Feature*** from the pop-up context menu. You are now in the line creation mode.

Click at the point where the highway enters the aerial image on the west side, then move the cursor to the point where the road curves, and click again to add a vertex. Continue tracing the road until you reach the intersection at the SW corner of the Town Square. At the point where you want to place your last vertex, right-click. Right-clicking ends line creation, and brings up the Modify Feature Info dialog box, allowing us to name the feature and specify its type.

In the Name box, type '89'. In the Feature Type drop-down, select Major/US Highway. Click OK to exit the dialog. Your screen should now look like this:



If you do not see black dots representing the vertices of your road, press ***Shift+V*** to toggle the vertex display mode.

Referencing the PDF map, notice that highway 89 heads north out of town at this intersection. We will draw another road segment for this piece of highway 89, then join the two.

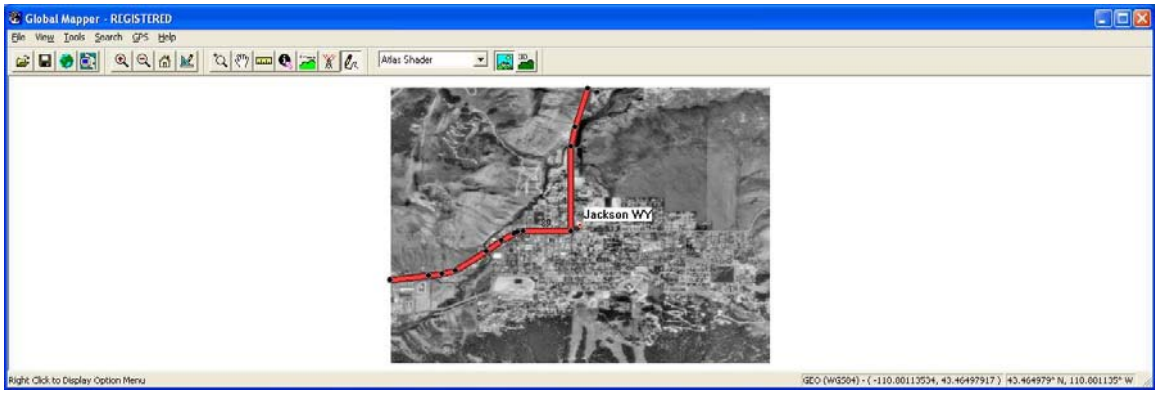
Position the cursor near the top of the screen where you completed the previous line segment. The cursor turn into an up arrow, indicating that by clicking you will scroll the screen. Click a few times until our Highway 89 road feature is at the bottom of the screen. If you haven't pressed ***Esc*** or switched to another mode, you are still in Line creation mode.

Click at the last vertex you drew, then begin a new line going north out of town.

> ***Advanced user tip***: *If the cursor turns into a down arrow at the point you want to click, press and hold the Shift key while clicking to override the scroll function.*

Move the cursor up to the point where the road curves NE, and click to add a vertex. The road will continue off the screen, but your cursor will change to an up arrow. Click while the up arrow is shown, and the display will redraw, leaving you in the line creation mode. Continue drawing the road until you get to the edge of the aerial image. For your last point, right-click to terminate the line. Name the road '89', the same as the other road segment. The same road type you specified for the first road segment, 'Major/US Highway' is already specified, so simply click OK to complete describing the Feature Info.

Press ***Home*** to show your entire project. Your screen should look like this:



Press ***Alt+D*** to enter Edit mode. Click the first highway 89 segment you created, then hold the ***Ctrl*** key while clicking the second segment to create a multiple selection. Right-click and select ***Combine Selected Line Features***. If a dialog pops up informing you of potentially conflicting feature values, click YES to combine features. Press ***Esc*** to exit Edit Mode. Your single line feature for highway 89 through Jackson WY is now complete. Double-clicking the line brings up the Modify Feature Info dialog if you want to make any changes after the feature is defined.

> ***A brief discussion about labels***
> *Your Garmin GPS receiver (GPSr) has the ability to display road names in upper and lower case, or use accented characters … but not both at the same time. It is important to understand this constraint early in your map design process. This constraint does not apply to city names or area features, only line features like roads and boundaries. If you are quite certain that you will not need an international character set in your map, then you may use upper and lower case in your road names. By default, GM8 creates a header specifying to cGPSMapper that you are using the international character set, meaning that only upper case letters may be used for street and road names. For now, I recommend that you accept this default behavior, meaning that your street names should all be specified in CAPS.*

Now we'll add some local streets. In the PDF map, zoom into the area around the Town Square. Note that the street continuing west from where highway 89 turns north is named Broadway. In GM8, press ***Alt+Z*** to enter Zoom Mode, then zoom to a similar area as shown in the PDF map. Press ***Alt+D*** to enter Edit Mode, then right-click to ***Create New Line Feature***. Click on the vertex where Highway 89 turns north, then create a line extending east to the edge of the aerial image. When you get to the right edge of the screen, instead of clicking the right arrow cursor to scroll the image, press the right arrow key on your keyboard to scroll the screen. Either method works equally well. Right-click to terminate Broadway at the east edge of town where the road turns north.

In the Modify Feature Info dialog, enter 'BROADWAY' as the Name, and select 'Residential Road' as the Feature Type from the drop-down list. You may object that Broadway doesn't really look like a residential street, but we're not really concerned with

making a distinction between types of city streets at this time. The key difference at this point is highways vs. city streets. 'Residential Road' is the built-in Global Mapper type matching the internal Garmin road types for city streets.

On the PDF reference map, note that Cache Street extends south from the Broadway-89 intersection, continuing past the baseball field until it stops. Add CACHE ST as a Residential Road.

The street running east-west below Broadway is called Pearl. It begins at Highway 89 on the west end, and terminates just past Gros Ventre Street on the east. Add PEARL AVE as a line feature. If you're not exactly sure where the road ends, don't worry about it. Welcome to digital cartography.

Let's add a few more streets and then we'll switch our attention to Point and Area features. Add DELONEY AVE running east-west just north of Broadway/89, and GILL AVE just north of Deloney. Just east of the Town Square add CENTER ST, running N-S between Gill and Broadway.

JACKSON ST runs north-south starting near the west end of Gill, south to Karns Ave at the NE corner of the fairgrounds. MILLWARD ST is the next N-S street to the east of Jackson. Note that at its southern end, Millward jogs slightly west before terminating, a detail which is not shown on the tourist map. Instead of following the line exactly, click once to define the northern end of Millward, skip over the jog and right-click to add your endpoint at the southern end of Millward. We'll fix up the jog next.

Press ***Alt+Z*** and click-drag to zoom into the area of the jog. Press ***Alt+D*** to enter Edit Mode, and click on Millward.

Right-click, then select ***Insert Vertex Into Selected Line***. Click at the corner of the jog. GM8 inserts a new vertex into Millward at the NE turn. Repeat at the SW turn, and you've fine-tuned Millward to match the aerial imagery.

Add FLAT CREEK DR running south from 89 on the west edge of the fairgrounds.

Add SNOW KING AVE running east from Flat Creek Dr past the southern edge of the fairgrounds, ending at Vine St at the west edge of the Snow King Resort.

Create VINE ST running north from Snow King Ave, curving NE then N, ending at Kelly Ave.

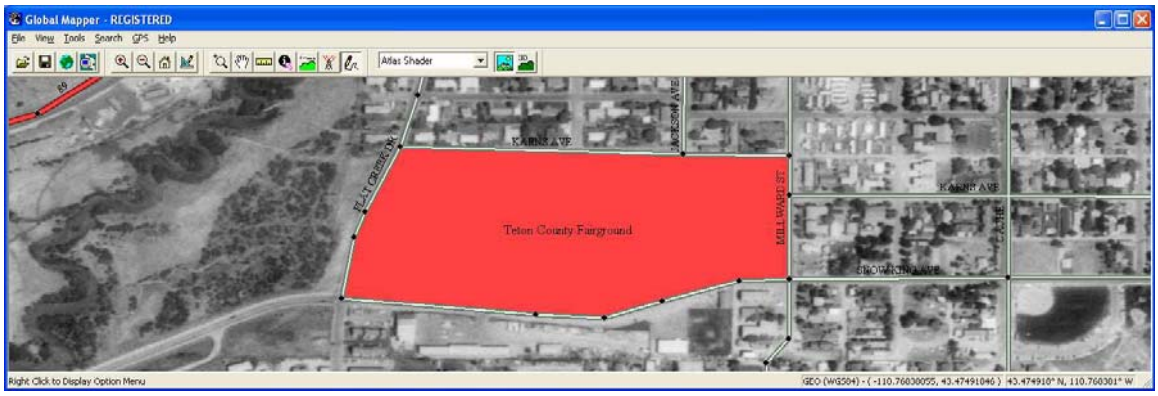
Karns runs E-W on the north edge of the fairgrounds. Note that it stops at Millward, then continues west one-half block south of that intersection. We'll define Karns in two sections. Define the first section of KARNS AVE from Flat Creek Dr to Millward. Define the second section of KARNS AVE from Millward east to Vine St.

An important note about road segments
Earlier we defined Highway 89 in two segments and joined them into a single line feature. Now we have Karns in two pieces, but we can't join them, because the street that connects them is already defined as Millward. It is important that any given road segment has only one name. This will be important when we add routing information. So think of it in the way that you would give directions to someone – "head east on Karns until it tees into Millward, turn right, then in half a block run left onto the continuation of Karns".

Let's switch our attention to some area features. We started with the Teton County Fairgrounds as our reference, so let's make it an Area Feature.

Press ***Alt+D*** to enter Edit Mode. Right-click to select ***Create New Area Feature***. Click your first point at the NW corner of the fairgrounds, at the corner of Flat Creek and Karns. Click on each of your vertices in the roads bounding the fairground, heading clockwise around the fairgrounds. Note that as your cursor approaches the existing vertices, it will snap to that location, making it easier to align your area image with existing features. Continue clicking clockwise until there is only one segment left, just south of the intersection where you began. Right-click to complete the Area Feature boundary.

In the Modify Feature Info dialog, name this feature Teton County Fairgrounds, and select 'Misc. Manmade Structure' as the Feature Type.



Miller Park is at the corner of Jackson and Gill. Beginning at the NW corner, click around clockwise to the first three corners to define the boundaries of Miller Park. GM8 will snap to the intersections making it easy to align the Area Feature with your existing Line Features.

Add the Town Square as a City Park.

Now let's add some Point Features.

Press ***Alt+D*** for Edit Mode, then right-click to select ***Add New Point/Text Feature***. In the center of the Town Square, click to add a Point Feature called Jackson, and assign its Feature Type to 'City, 10k-50k'.

Albertson's is south of the Highway 89 – Flat Creek Dr intersection. Click in the center of the large white building at this intersection to create a Point Feature. Name the feature "Albertson's" and assign its Feature Type to 'Shopping'. Press ***Alt+G*** to enter Grab Mode, then click and drag your map to the right to show the area west of Albertson's and south of Highway 89.

Add the Point features Grand Teton Plaza and Powderhorn Mall, both with Feature Types of 'Shopping'.

Add the Virginian RV Park with a Feature Type of 'Lodging'. Add the Cowboy Village Resort and the Snow King Lodge as lodging features.

Our project is beginning to take shape. Let's look at the structure of the project and save it.

Maximize your GM8 screen, then press ***Home*** to show the full project in your display. Press ***Alt+C*** to open the Overlay Control Center. You will see that there are two overlays in our project, the TerraServer DOQ aerial imagery, and 'USER CREATED FEATURES'. User Created Features are everything that we've added to the map so far – Line Features, Area Features and Point Features.

Select the TerraServer layer, then click ***Hide Overlay*** to see just your User Created Features. Click the same button, now labeled ***Show Overlay***, to show the aerial imagery again.

Click Close to dismiss the Overlay Control Center.

In the File Menu, click ***File | Save Workspace As***. Navigate to your \SAMPLE MAP PROJECT\SOURCE\CITIES\JACKSON WY folder. In the File name box, type JACKSON WY, then Save. Your project workspace is now saved as JACKSON WY.GMW. The .GMW extension means 'Global Mapper Workspace'.

Click ***Ctrl+U***, which unloads the GM8 workspace.

To reload your project, click ***File | Load Workspace***, or at the bottom of the File Menu, in the recently used file list, click JACKSON WY.GMW.

If your internet connection is slow or you'll be working offline, you will want to have a local copy of the downloaded TerraServer data on your machine. Let's save a copy now.

In the File Menu, click ***File | Export Raster and Elevation Data***, then select ***GeoTIFF***. In the GeoTIFF Export Options dialog, select 8-bit Palette Image (PackBits/LZW

Compressed). This format offers the smallest file size with the best performance in GM8. Accept the default settings, and click OK. Save it in your \SAMPLE MAP PROJECT\SOURCE\CITIES\JACKSON WY\ folder as JACKSON WY.TIF. This export will take a few moments. When the save is complete, we can reload our project with the local aerial imagery rather than the live online data.

Press ***Alt+C*** to open the Overlay Control Center. Select the TerraServer overlay to highlight it, then click ***Close Overlay***. Close the Overlay Control Center dialog. Press ***Ctrl+O*** to bring up the File Open dialog. Navigate to the \SAMPLE MAP PROJECT\SOURCE\CITIES\JACKSON WY\ folder. In the 'Files of Type' dropdown box, select ***Supported Commonly Used Types***. Select JACKSON WY.TIF. The local copy of the aerial image loads. Press ***Alt+C*** to open the Overlay Control Center, and you'll now see that JACKSON WY.TIF is shown as the first overlay. Press ***Ctrl+S*** to save your workspace using your local copy of the TerraServer aerial image. Press ***Home*** to zoom your workspace to show the entire project.

Now let's have some fun and show off some of GM8's real power. Click ***File | Download Online Imagery/Topo Maps***. In the Select Online Data Source to Download dialog, select ***United States Elevation Data (NED) (30m Resolution)***. In the Select Area to Download section, select ***Current Screen Bounds***. The elevation data loads on top of the aerial imagery, but behind the User Created Features. Let's move this overlay to display behind the aerial imagery. Press ***Alt+C*** to open the Overlay Control Center, then right-click the Elevation Data overlay. Select ***Move Selected Overlays to Top of List (Draw First)***. Click Close to exit the Overlay Control Center.

In the menu bar, click the 3D button 

at the right edge of the control icons. GM8 renders a perspective 3D view of your project, combing the elevation data we just downloaded with the aerial imagery and our User Created Features draped on top. Click and drag in the 3D view to rotate the image. Pretty cool, eh?

Later in the tutorial we'll use this elevation data to create contour lines in our map. But first, let's save the data for offline access.

You may have noticed that the loaded elevation data is slightly larger than our aerial imagery, due to the different tiling sizes of the online data sources. We can crop the data set to match the aerial imagery before saving it.

Open the Overlay Control Center, then right-click on JACKSON WY.TIF. Select ***Create Area Features from the Selected Layer Bounds***. Close the Overlay Control Center, and double-click in an open area of the aerial image. The Modify Feature Info dialog comes up for the Area Feature you just created. Note that it is auto-named to be the same as the feature it was created from, and that its Feature Type is Coverage/Quad. Click OK to dismiss the dialog. The Area is still selected, as indicated by a cross-hatch pattern.

Click ***File | Export Raster and Elevation Data***, then select ***Export Global Mapper Grid***. Select Feet as the Vertical Units, accepting all other default selections on the General tab.

Click on the Export Bounds tab, and select ***Crop to Selected Area Feature(s)***, and then OK. Save the file as JACKSON WY.GMG in the same directory. You can now close the live online overlay and reload your cropped .GMG file.

We don't need the Area Feature covering the aerial imagery anymore, so let's get rid of it. In the menu bar, click ***Search | Find By Name***. Unclick 'Lines' and 'Points' so that only Areas are shown in the list. Select JACKSON WY.TIF then click ***Delete Selected***. Close the Find dialog, and save your project.

Compiling your data

Next we will export our vector data in the USER CREATED FEATURES overlay in preparation for compiling it with cGPSMapper. In the File menu, select ***File | Export Vector Data | Export Polish MP (cGPSMapper) File***. In the Polish MP Export Options, type 'Jackson WY' in the Map Name box, your name in the Copyright box, and accept all other default options. Save the Polish format file as JACKSON WY.MP in the same directory as your other data files for this project.

Now we'll compile our exported .MP file. cGPSMapper is a command line program, meaning that is launched from a command shell. The simplest way to do this is to use a batch file, which we'll create now.

By default, cGPSMapper is installed at C:\PROGRAM FILES\CGPSMAPPER\CGPSMAPPER.EXE. The batch file we're creating depends upon this location – if your installed location is different, adjust as necessary.

Open your text editor. Create the following single line, with the quotation marks:

```
"C:\Program Files\cGPSMapper\cGPSMapper.exe" "Jackson WY.MP" -o
"Jackson WY.img"
```

Save this file in your data directory as MAKE JACKSON WY.BAT.

Open an instance of Windows Explorer (you can use the system shortcut keys ⊞+E). Navigate to your data directory \SAMPLE MAP PROJECT\SOURCE\CITIES\JACKSON WY\, and double-click the MAKE JACKSON WY.BAT file that you just created. cGPSMapper will launch with JACKSON WY.MP as its input, creating JACKSON WY.IMG as its output.

Downloading your data with Sendmap20

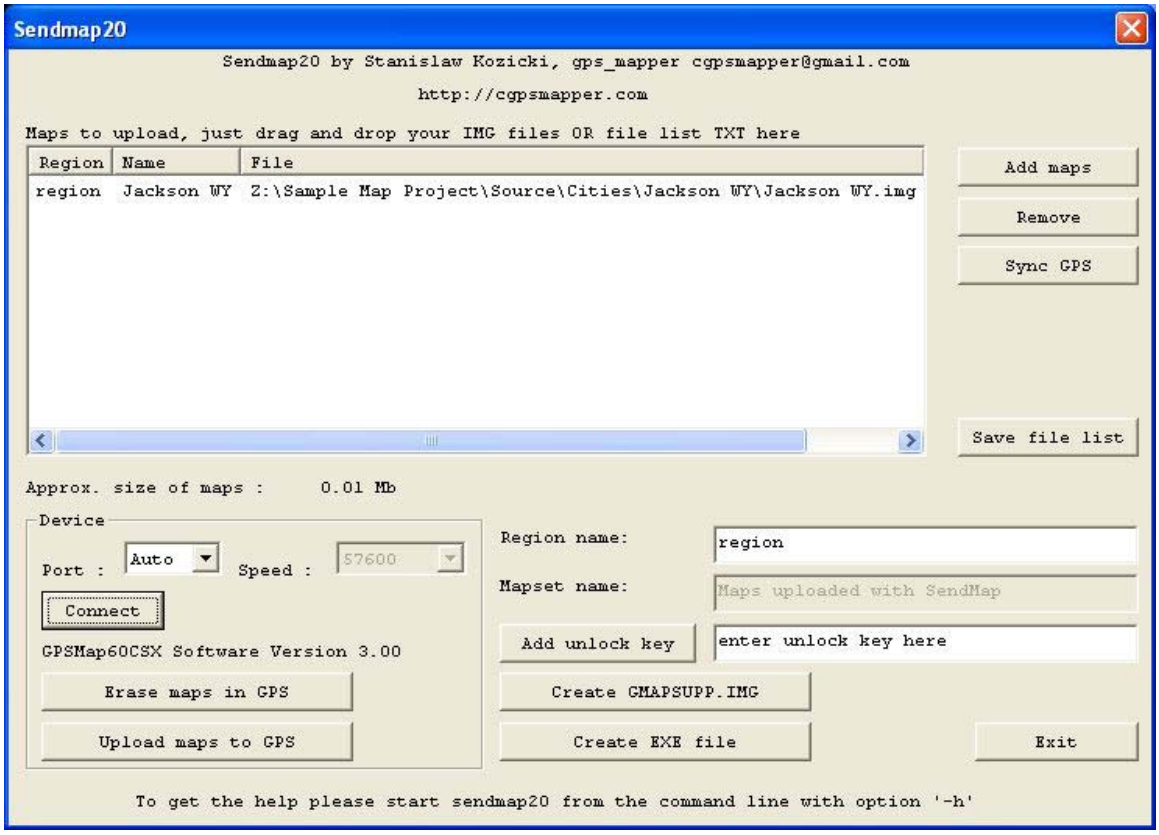
We'll create another batch file to launch SendMap20, another program from cGPSMapper which downloads compiled .IMG files to your GPSr.

Open your text editor and create the following line (with quotation marks):

```
"C:\Program Files\cGPSMapper\sendMap20.exe"
```

Save this file in your data directory as LAUNCH SENDMAP.BAT.

Open an instance of Windows Explorer. Navigate to your data directory \SAMPLE MAP PROJECT\SOURCE\CITIES\JACKSON WY\. Double-click the LAUNCH SENDMAP.BAT file that you just created. Click the Add Maps button, navigate to \SAMPLE MAP PROJECT\SOURCE\CITIES\JACKSON WY.IMG, and select it for uploading.



Depending on your model of GPSr, you will either upload your map directly, or make a GMAPSUPP.IMG file and use the Windows file system to copy it to your GPSr.

60CSx	nüvi
1. Select maps for uploading. 2. Connect GPSr to computer. 3. Turn GPSr on. 4. Click Connect button. 5. Wait for GPSr description to appear under Connect button. 6. Click 'Upload maps to GPS'. 7. View map.	1. Select maps for uploading. 2. Click 'Create GMAPSUPP.IMG'. 3. Connect nüvi to computer. 4. Turn nüvi on. 5. Open Windows Explorer (⊞+E). 6. Navigate to GMAPSUPP.IMG, select, Ctrl+C to copy file. 7. Navigate to \Garmin Nuvi\Garmin directory, Ctrl+V to paste. 8. In the system tray, click 'Safely Remove Hardware' and select drive representing nüvi device. 9. Your nüvi reboots. 10. View map.

This completes the first tutorial.

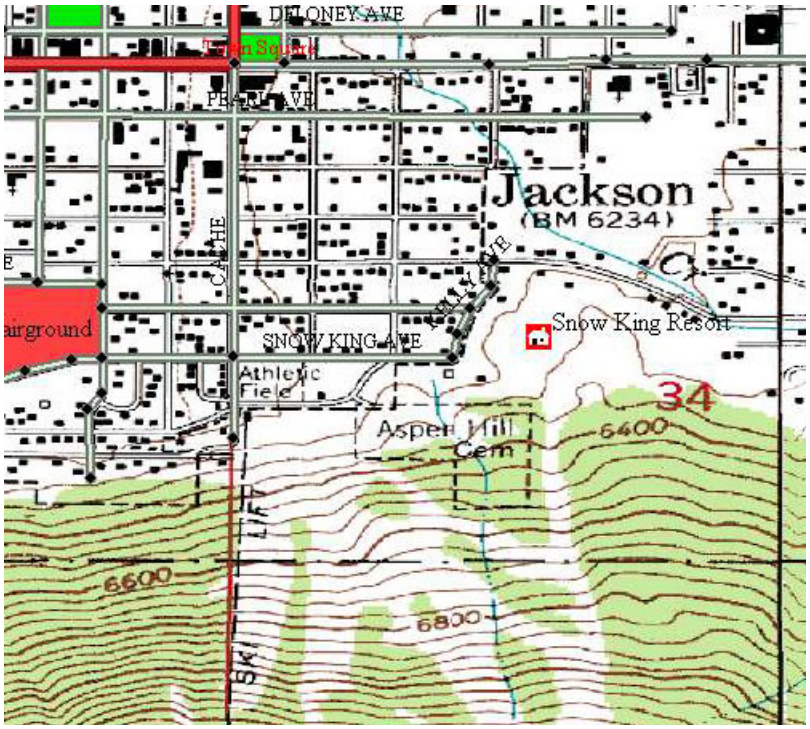
Tutorial 1b – Generating Contours

In this tutorial we will use the downloaded Elevation Data to create topographic contour lines for Jackson.

By convention, printed topographic maps use thick lines to represent 'Index Contours', medium lines to represent 'Intermediate Contours', and dashed lines to represent 'Supplementary Contours'. GM8 supports this convention.

In the Overlay Control Center, right-click on JACKSON WY.GMG and select ***Zoom to Selected Layer(s)*** to fill the screen with our topographic data, then close the Overlay Control Center.

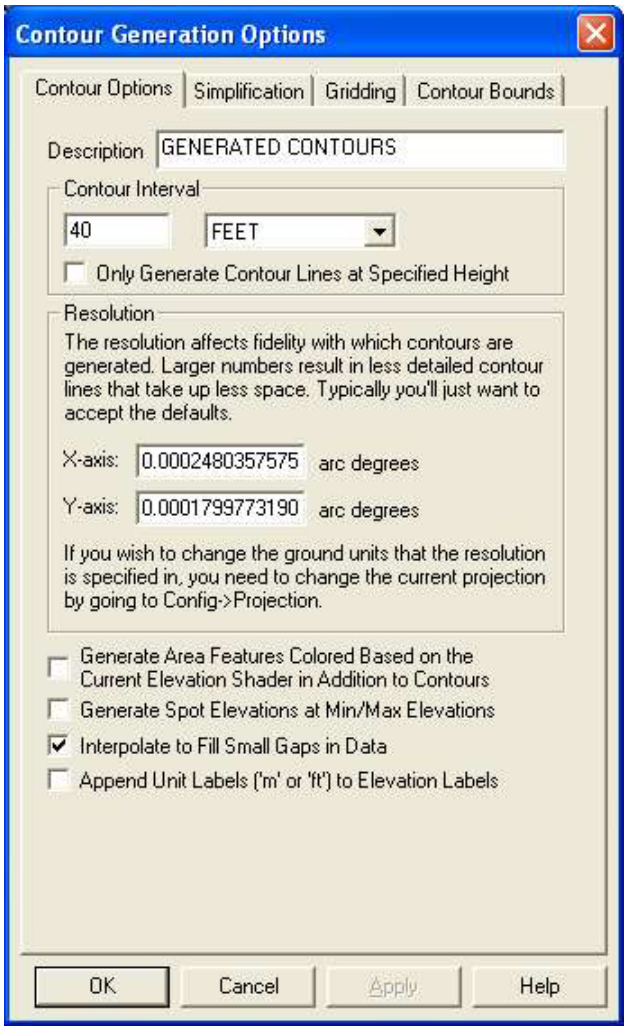
Go to ***File | Download Online Imagery/Topo Maps***, and select the 'DRG – USGS Digital Raster Graphics (Topographic Maps) option, with ***Current Screen Bounds*** as the Area to Download. GM8 will retrieve a scanned topographic map of our area of interest and add it as an overlay in our workspace.



Note that the thick, labeled contour lines are spaced every 200' at 6400', 6600', and 6800'. By convention, USGS maps have four *intermediate contour* lines between each *index contour* or *major* contour line. In this area, the intermediate contours are spaced at 40'. There are slight differences in the ways that contour lines are named, but they are all equivalent.

USGS	GM8	cGPSMapper
Index Contour	Contour line, Major	Land Contour (thick) Type=0x22
Intermediate Contour	Contour line, Intermediate	Land Contour (medium) Type=0x21
Supplementary Contour	Contour line, Minor	Land Contour (thin) Type=0x20

Let's use GM8 to generate contours matching the USGS topographic map.
We want our intermediate contours to fall at 40' spacing. Click ***File | Generate Contours…*** change Contour Interval to 40', and uncheck the 'Append Unit Labels ('m' or 'ft') to Elevation Labels' checkbox.



Click OK. GM8 generates the contour line set every 40', with intermediate and major contours alternating every 200' closely matching the USGS map.

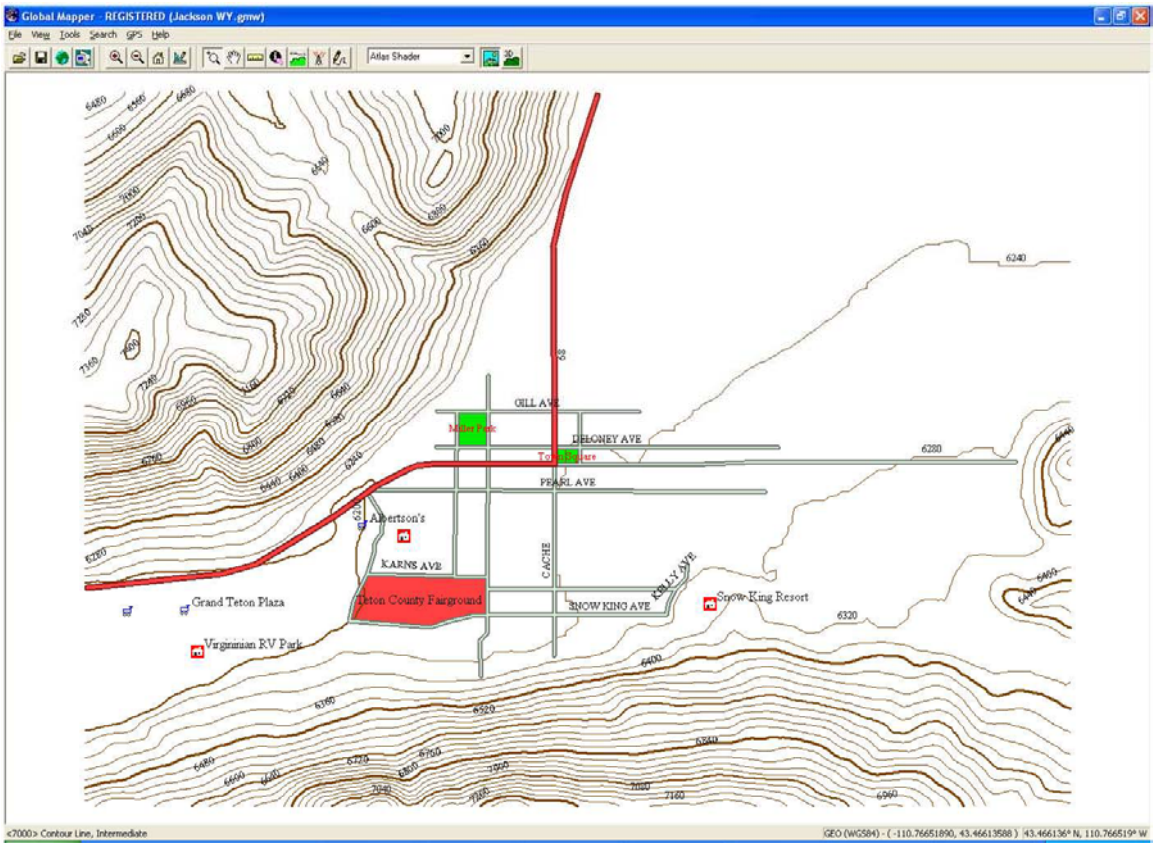
> ***Advanced User Tip***: *To toggle the display of the vertices, press* **Shift+V**.

Exporting our combined data set

When we generated the topographic contour lines, GM8 created a new overlay called GENERATED CONTOURS. This is a *vector* overlay, just like our USER CREATED FEATURES overlay. Vector overlays (Points, Lines and Areas) are the only data that can be compiled by cGPSMapper and downloaded to your GPSr. *Raster* overlays (like the USGS elevation map, or the TerraServer aerial imagery) is only used for reference when digitizing, or making printed maps.

We now have two vector layers –USER CREATED FEATURES, with our roads, points and areas, and GENERATED CONTOURS, with our topographic contour lines. Let's do another export for compilation. Open the Control Center, and hide the non-vector layers – JACKSON WY.GMG, JACKSON WY.TIF, and TERRASERVER DRG (JACKSON, WYOMING, UNITED STATES).

We are now seeing only the data that we've created for our map. It is not necessary to hide the raster layers when exporting, but it's a good habit to get into to be sure that what you're exporting is what you want in your data file. Your screen should look something like this:



From the menu, select ***File | Export Vector Data | Export Polish MP (cGPSMapper) file***, and specify JACKSON WY as the Map Name, then select JACKSON WY.MP as the filename, and click OK. GM8 will ask you to confirm that we are overwriting the previous version of the file. Click OK.

Double-click your MAKE JACKSON WY.BAT file to invoke cGPSMapper to compile our latest source file. Notice that it takes a bit longer to process the contour lines. Download the .IMG file to your GPSr using SendMap20, and enjoy the fruits of your labor.

Tutorial 2a – Controlling Object Visibility

Goals for this section

- A first look at Polish format source code
- Understanding object visibility
- Creating a Template File

To this point, we have been working with general defaults in creating our map. You now know enough of the basics that if you want to start working on your own map, you can now do so. The following tutorials in this document discusses Object Visibility, Custom Types and Routing. You can come back to rejoin the tutorials at any time – nothing we introduce subsequently will cause you to create or manage your data differently.

A First Look at Polish Format

As you now know, Polish format (.MP files) is the name for the source code that cGPSMapper converts to Garmin executable format (.IMG). Historically, it's called Polish format because the author of cGPSMapper is from Poland, and when you write a great program like cGPSMapper, you get to name things.

An .MP file can be very simple, with a single Point object declaration, or it can be very complex, with thousands of object declarations. For a full description of cGPSMapper compiler functions, go to the cGPSMapper website for the latest documentation at http://cgpsmapper.com/manual.htm.

At a minimum, an .MP file has a *header* declaration, which tells the compiler a few basic things about our map, and one or more *object* declarations. Declarations in Polish format are always of the following form:

```
[declaration]
Attributes
[End-declaration]
```

When you tell GM8 to export a .MP file by using the ***File | Export Vector Data | Export Polish MP (cGPSMapper) File*** command, GM8 by default will create a header declaration, plus object declarations for your created features.
The default header generated by GM8 looks like this:

```
[IMG ID]
ID=423758661
Name=Jackson WY
Elevation=F
LBLcoding=9
Codepage=1252
Marine=N
Copyright=Mike Mapmaker
Preprocess=F
;TreSize=3000
POIIndex=Y
```

```
Transparent=N
Levels=4
Level0=24
Level1=22
Level2=20
Level3=18
Zoom0=1
Zoom1=2
Zoom2=3
Zoom3=4
[END-IMG ID]
```

For a full explanation of all the header declaration attributes, consult the cGPSMapper manual. For the purposes of this tutorial, we'll look specifically at only a few lines dealing with object visibility.

Understanding Object Visibility

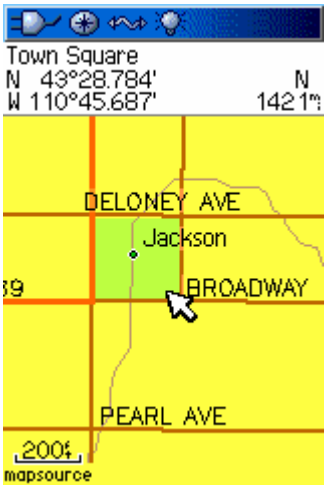
Note the lines from the `[IMGID]` header referencing Levels:

```
Levels=4
Level0=24
Level1=22
Level2=20
Level3=18
```

These lines tell cGPSMapper that our map will be defined with 4 levels of zoom detail. The lowest level, or smallest scale zoom, is Level0. '24' refers to the precision of the latitude and longitude coordinates provided for the object. When declaring the location of objects in the world, the highest precision available uses 24 bits of resolution, which resolves to an accuracy of about 2.5 meters. Consumer GPS units have a best-case accuracy of about 8 meters. The header attribute of `Level0=24` means that objects declared at level 0 will be stored with 24-bit accuracy, or a location of +- 2.5m.

Let's look at how this works with the simplest object declaration, a Point declaration for the town of Jackson WY:

```
[RGN20]
Type=0x09
Label=Jackson
Levels=2
Data0=(43.480039,-110.7618274)
[END]
```



- **`[RGN20]`** begins an object declaration for a Point Feature.
- **`Type=0x09`** is an attribute declaring that this Point Features is a 'City, 10k-50k'.
- **`Label=Jackson`** corresponds to the name we provided when creating the Point Feature.
- **`Levels=2`** means that the object will be visible for 2 zoom levels.
- **`Data0=`** tells cGPSMapper where in the world Jackson is located using its longitude (N/S) coordinate, followed by its latitude (E/W) coordinate. Coordinates in the western and southern hemispheres are negative, while coordinates in the northern and eastern hemispheres are positive. **`Data0`** also asserts that this object is defined to exist at Level0 as its most detailed level, so cGPSMapper will store 24 bits of location information for this object's coordinates.
- The **`[END]`** statement indicates that this object declaration is complete.

When cGPSMapper compiles this Polish format source code, it will create a pair of 24-bit coordinates for use when displaying objects at the most detailed zoom level, and a pair of 22-bit coordinates for use when displaying this object at Level1. This is significant because an object whose visibility spans multiple levels must have a coordinate pair in the data file for each level of visibility. By using less and less precision at the lower zoom levels, less storage space is required, making the file format more compact and efficient. cGPSMapper handles all of this for us during compilation, but it's an important factor contributing to file size and map accuracy.

For our purposes in this tutorial, the key concept to grasp here is that GM8 is creating a default range of visibility for our objects, based upon the object type.

Controlling Object Visibility in the GPSr

If you've played with your GPS receiver, you know that you can influence how and when objects are displayed. Depending upon the Garmin model, there are settings in the menu associated with the map display page allowing you to increase or decrease map detail and whether or not certain object types are displayed, and how their labels are shown. These end-user controls are beyond the control of the mapmaker – there's nothing we can do at map creation time that will override local settings that the user chooses at map display time. During map development, I recommend maintaining the default display settings for your reference GPSr so you have a pretty good idea of what the average user will experience.

Specifying Object Visibility in the Map Data

We can use GM8 and cGPSMapper to help us control how and when objects are displayed. GM8 automatically assigns default visibility attributes when you create objects. For example, Major Highways will be assigned a higher **`Levels=`** attribute than a Residential Road, meaning it will be visible at lower zoom magnification levels than residential streets.

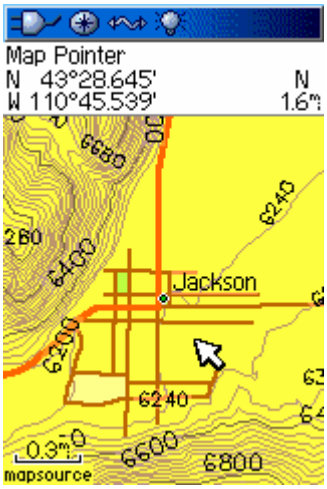
Here is the exported Polish format code for Highway 89 in our map. Note the **`Levels=2`** statement:

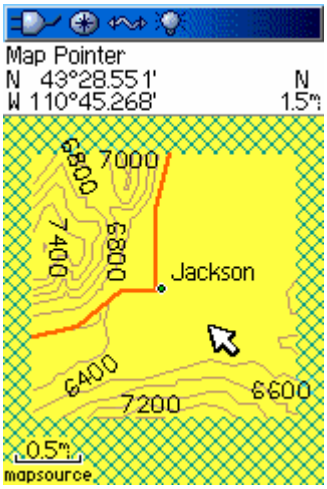
```
[RGN40]
Type=0x02
Label=89
Levels=2
Data0=(43.4745104,-110.7817081),(43.4749766,-
110.7775745),(43.4751009,-110.7761759),(43.4755049,-
110.7747152),(43.4775562,-110.7713897),(43.4787372,-
110.7697424),(43.479421,-110.7684682),(43.4795764,-
110.7680641),(43.4796696,-110.7674425),(43.4796696,-
110.7623144),(43.4887526,-110.7623766),(43.4908894,-
110.7618793),(43.495023,-110.7605429)
[END]
```

In this declaration for Broadway, note the `Levels=1` statement:

```
[RGN40]
Type=0x06
Label=BROADWAY
Levels=1
Data0=(43.4796696,-110.7623144),(43.4796075,-
110.7568909),(43.479723,-110.7544272),(43.479723,-
110.7522714),(43.479723,-110.7499039),(43.4797615,-110.7431864)
[END]
```

Here's our map zoomed to a scale of 0.3mi and 0.5mi:





As we zoom out to a scale of 0.5miles, Highway 89, the city dot and the major contour lines remain visible as Level 2 objects, but the city streets and minor contours – Level 1 objects -- have disappeared.

GM8 knows all of the standard Garmin data types and how to assign the correct visibility attributes to each of them when creating your features, so for most mapmaking, you may never need to make changes to the default settings.

Using a Template File

The purpose of a Template File is to provide GM8 with a specific header that you want to in place of GM8 creating a header automatically for you at export time. There are several reasons you may wish to use your own header:

- Specifying your Map Name and Copyright so you don't have to retype it each time you export
- Managing the Levels in your map
- Managing advanced attributes like Routing

We will create a Template File with GM8's assistance, then use it to fill in our Map Name and Copyright automatically.

Using your text editor, open your exported JACKSON WY.MP.

Highlight the section from `[IMG ID]` through `[END-IMG ID]` at the beginning of the file, then press ***Ctrl-C*** to copy it to the Windows clipboard.

Create a new file, and paste the `[IMG ID]` header from the clipboard into the new file window.

In the `ID=` attribute, delete the number after the equal sign so that the line simply reads `ID=`. GM8 will automatically generate a unique number for each map created at export time, so there's no need to specify a map ID number.

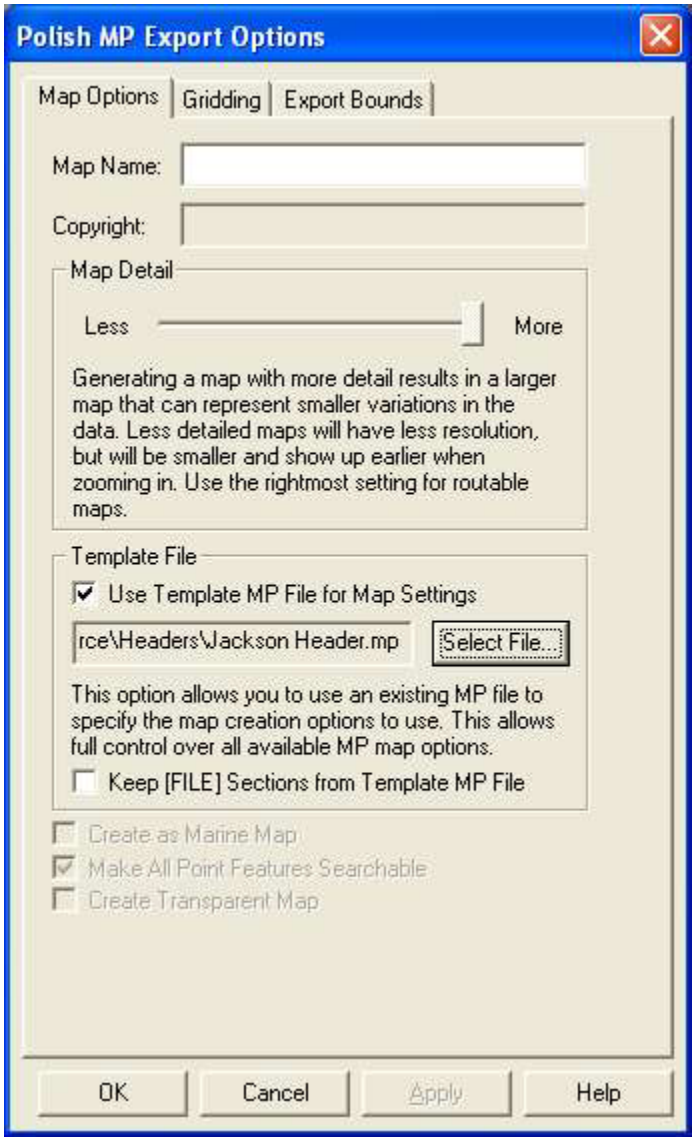
In the Copyright attribute, add your name.

Your Template File should now look something like this:

```
[IMG ID]
ID=
Name=Jackson WY
Elevation=F
LBLcoding=9
Codepage=1252
Marine=N
Copyright=Mike Mapmaker
Preprocess=F
;TreSize=3000
POIIndex=Y
Transparent=N
Levels=4
Level0=24
Level1=22
Level2=20
Level3=18
Zoom0=1
Zoom1=2
Zoom2=3
Zoom3=4
[END-IMG ID]
```

Save this file as \SAMPLE MAP PROJECT\HEADERS\JACKSON HEADER.MP, then exit your text editor.

Return to GM8. In the File menu, select ***File | Export Vector Data | Export Polish MP (cGPSMapper) File***, bringing up the Polish MP Export Options dialog.



In the Template File section, click the checkbox to 'Use Template MP File for Map Settings'. Press the ***Select File*** button to locate the JACKSON HEADER.MP file you just created in your \SAMPLE MAP PROJECT\SOURCE\HEADERS\ directory.

Click OK to export your file. Open it with your text editor, and confirm that your Map Name and Copyright attributes were transferred from your Template File.

An Important Note about Template Files created by GM8

We used GM8 to create our Template File, based on our currently defined Features. The `Levelx=` *attributes were generated by GM8 as a function of the current object database. Different objects are visible up through different levels. For example, Interstate highways and oceans are visible several levels above State Highways or streams. If you make significant changes to your map, you will need to create a new template file to properly see your new features on your GPSr.*

Tutorial 2b – Custom Types in GM8

Goals for this section

- Understanding Custom Types in GM8
- Understanding Custom Types in cGPSMapper

So far we've downloaded data from the Internet, digitized some roads and local features, generated topographic contour data, and downloaded iterations of our map into our GPSr. The next area we'll address is Custom Types.

Custom types allow you to control the appearance of your on-screen data. Custom types can be as simple as changing the representation in GM8 of how a road looks, or as complex as changing the internal representation of a Garmin data type in your GPSr. We'll cover the entire range of possibilities in this section.

Let's begin with altering the look of displayed data in GM8. Each standard type in GM8 has a default visual representation used when rendering that type onscreen. For example, press ***Alt+D*** to enter Edit Mode in GM8, then double-click on Highway 89.

In the Modify Feature Info dialog, we have two choices for the Feature's rendering style:

- 'Use Default Style for the Selected Feature Type' (the current selection), or
- 'Specify Style to Use When Rendering Feature'.

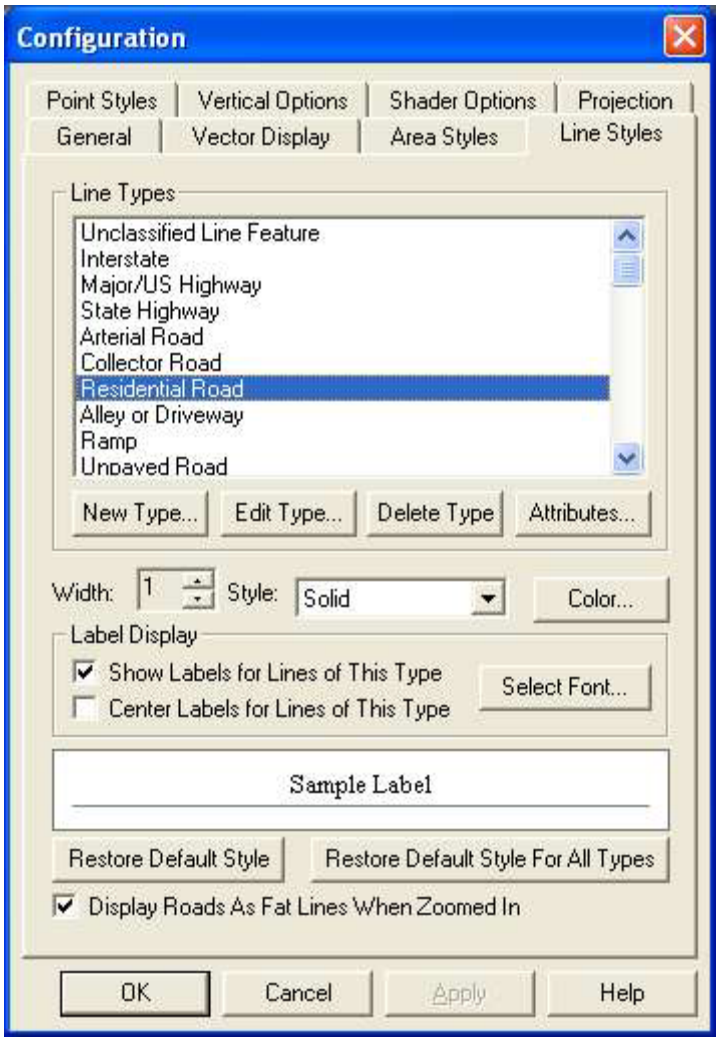
A style specified in this dialog is local to the selected Feature(s). This modified style is not globally available to use with other Features.

Click the 'Specify Style to Use When Rendering Feature' radio button, then click 'Customize Style'.

In this dialog, I've changed Width from 2 to 3, and Color from red to blue.
Click OK, and the changes are applied to the currently selected Feature(s). If you wish to revert to the standard rendering style for this line, simply reopen this dialog by double-clicking on the Feature, then click the Feature Style radio button labeled 'Use Default Style for Selected Feature Type'.

We can also change the default rendering for the default Feature Types. This will affect all objects of a given Feature type.

In the menu, select ***Tools | Configure…*** This brings up the Configuration dialog. Click the 'Line Styles' tab, then scroll to 'Residential Road'.



For built-in types like Residential Road, you can alter the line width, the color of the lines, and the appearance of the font used to label the Feature. Uncheck the 'Display Roads As Fat Lines When Zoomed In' checkbox, then try different styles for Residential Roads. Click the Apply button to see your changes. Note that all Residential Roads are affected by a change made in this dialog.

Finally, let's create a new Feature Type.

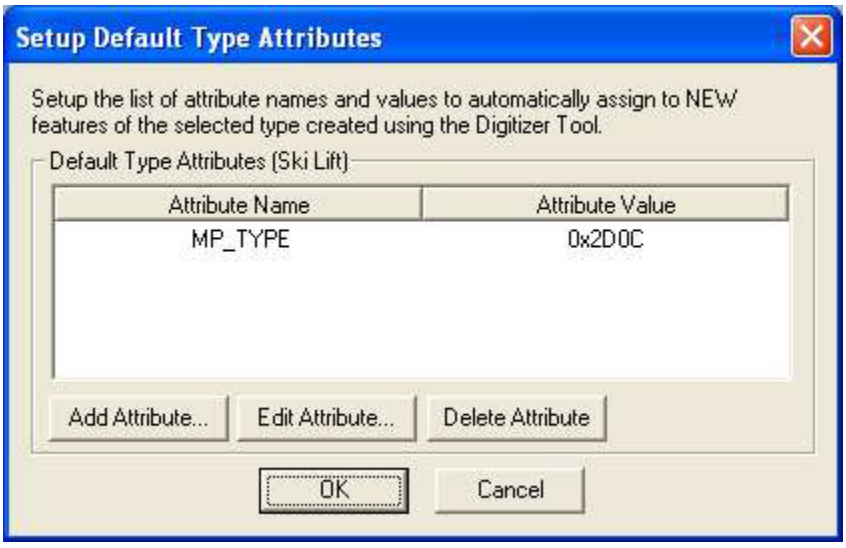
In the Configuration dialog, click the 'Point Styles' tab, then click 'New Type'. Enter 'Ski Lift' as the Type Name, and select 'Skiing' as the symbol.

Now we have a new custom Ski Lift feature available to use in our map, but cGPSMapper doesn't know anything about Ski Lifts. We need to provide some Type information for this object.

Consulting the cGPSMapper manual at http://cgpsmapper.com/manual.htm we see in section 8.3.1, POI Types, that objects of type 0x2Dxx are similar to our new Ski Lift

type. The last referenced object of this type listed in the manual is 0x2D0B, so let's take the next higher one in the series at 0x2D0C for our custom Ski Lift type.

With 'Ski Lift' selected in the Configuration dialog, click the 'Attributes' button to add a default attribute whenever objects of Type 'Ski Lift' are created.



Click 'Add Attribute…' and enter 'MP_TYPE' as the Attribute Name, and '0x2D0C' as the Attribute Value. Exit all the dialogs.

Zoom in near the area of the Snow King Resort. About two blocks left of the Snow King Resort in Phil Baux Park, just SE of the baseball diamond is the ski lift. In Edit Mode, right-click to ***Add New Point/Text Feature***, then click at the location of the lift and create Phil Baux Park as a new Point Feature of type Ski Lift.

> ***What Undefined Type Values are Safe to Use?***
> *If your map will be used in conjunction with a commercially available map, then you should pay attention to the standard data types by checking the cGPSMapper manual before adding your own data type. The worst that could happen is that there are other objects sharing the same type in the other maps loaded on your GPSr, which could cause some confusion.*
>
> *If your map is intended to be used alone, without depending on a Garmin basemap or other user-loaded maps, then you have much more latitude in using Type IDs as you wish. However, it's a good idea to stay close to the existing scheme in case you want to integrate your map with another in the future. Scan the various pre-assigned types in the cGPSMapper manual so you're familiar with the general categories.*

Custom Types in your GPSr

So far we've discussed modifying the representation of your objects within GM8 on the computer. It's also possible to modify the representation of objects in your GPSr. This

capability is limited to recent Garmin models. At this time, GM8 does not support the full range of possible presentation styles available in the GPSr, but you can get pretty close.

Tutorial 2c – Creating Custom Types with cGPSMapper

Goals for this section

- Understand how custom type files are created for a compatible GPSr
- Create a simple custom type file with cGPSMapper
- Download a map with a custom type file to your GPSr

The source file for custom type definition is very similar to a conventional .MP Polish format source file, but with different declarations applying specifically to bitmap descriptions for your custom types, plus instructions on the draw order for area objects. We will refer to these source files as .MPT, or .MP type files. The compiled version of the file has a .TYP suffix.

At a minimum, your .MPT file will contain a `[_ID]` section, a `[_drawOrder]` section, and one or more custom definitions for objects – `[_point]`, `[_line]` or `[_area]`.

There is currently no GUI editor for .MPT files – you're going to have to roll up your sleeves and create your own using a text editor, graphics editor and reference materials. Let's get started.

[_ID] Section

```
[_ID]
ProductCode=1
FID=888
[END-ID]
```

This declaration goes first in your .MPT file. Its most important function is to tell cGPSMapper your Family or Product ID (FID), corresponding to a similar FID declaration in your Preview source file. This keeps your various map elements connected to one another in MapSource. 888 is a generic FID often used by beginning map makers; it's unlikely that you will have a collision with another map of this FID on your own system. If you are considering creating maps for distribution, there is an unofficial database and registry for FIDs at http://www.keenpeople.com/index.php?option=com_maplist&Itemid=78.

[_drawOrder] Section

At a minimum, your .MPT source file must define the draw order for ALL polygon types in your map, 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 0x08, priority 3) will be drawn 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]
```

Custom Type Definitions

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

[_point] Definitions

Points (POIs) define your replacement bitmap for the associated Point 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.

In addition, you may also specify up to four language strings defining the default label for the Point type. This label is 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.

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 use `Nightxpm=`, the `DayXPM=` or simply `XPM=` definition will be used for both day and nighttime rendering.

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 Point definition to illustrate 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 this definition to be 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 follows, followed by a space, ‘#’, then the hex RGB color value.

Following the color declarations is the bitmap description:

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

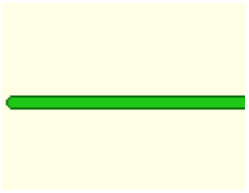
This XPM bitmap describes a 4x4 rectangle with a black 1-pixel border and a transparent center.

[_line] Definitions

Line definitions are used to replace the standard line 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.

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]
```



The `[_line]` definition above specifies a replacement rendering for lines of Type=0x01, a 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 statement indicates that there is no associated pixel bitmap, only color definitions. This is because we are using `LineWidth=` and `BorderWidth=` instead of a bitmap. 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.

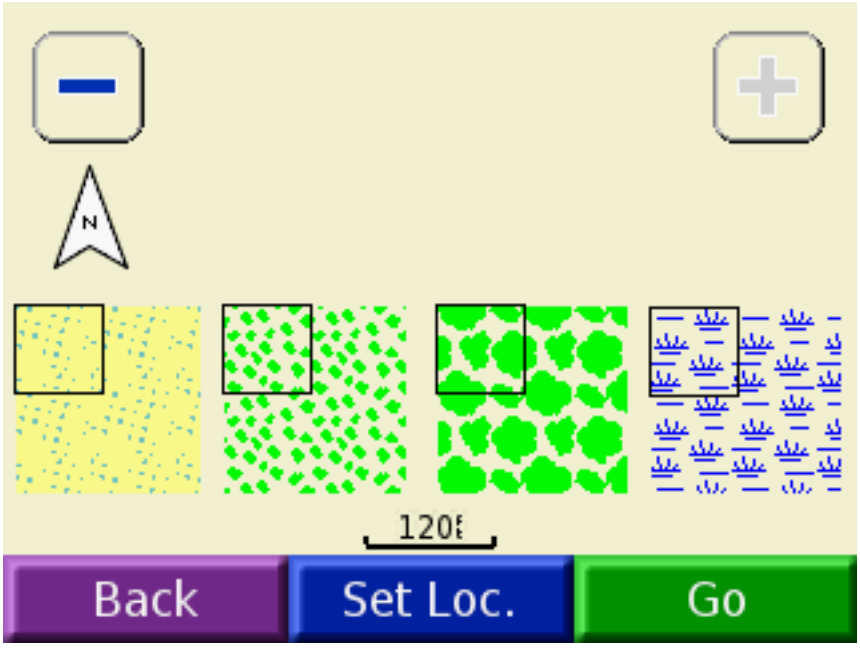
Method 2: Describe a bitmap using XPM:

```
[_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 this 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 the texture and color underneath the line.

[_polygon] Definitions

Polygon definitions are limited to 32x32 xpm bitmaps using of 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 tile on the left is a custom definition (described below), and the following three are standard types 0x4F, 0x50 and 0x51. The black boxes outline the 32x32 tiles. Notice that the leftmost tile uses two colors, while the next three tiles use one color plus transparency.

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]
```

Name substitution

You may create up to 4 default names associated with different languages to be displayed 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			

Creating XPM bitmaps

If you want to create anything other than the simplest shapes for your custom Points and Areas, you will want to use a graphics tools to manage your source bitmaps and output your XPM definitions. The following workflow description uses Photoshop Elements, IconXP and Microsoft Word. This is certainly not the only way, but it works.

Photoshop:

- 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 saved 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 and paste into your source file, starting with the quotation mark before the first line of the declaration, all the way to the closing brace.

Text Editor steps:

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

Putting it all together

- Create a .MPT file with your custom type definitions.
- Compile your custom type file with cGPSmapper, using the typ switch: cGPSmapper typ <YOURCUSTOMTYPES>.MPT
- Use Sendmap 2.0 to combine your .IMG file and your .TYP file into a single upload or GMAPSUPP.IMG.

Tutorial 3a – Routing

Goals for this section

- Understanding Routing
- Use GM8 to insert vertices at intersections
- Export .MP file to GPSMapEdit (GME)
- Create routing data in GPSMapEdit
- Use the Search feature in GM8 to edit multiple features simultaneously

Some Garmin GPS receivers have the ability to calculate routes from your current location to your destination, then provide turn guidance along the calculated route. If your GPSr has this capability, then you can create routable maps with cGPSMapper's Personal version. A free 30-day trial is available at http://cgpsmapper.com/buy.htm#personal.

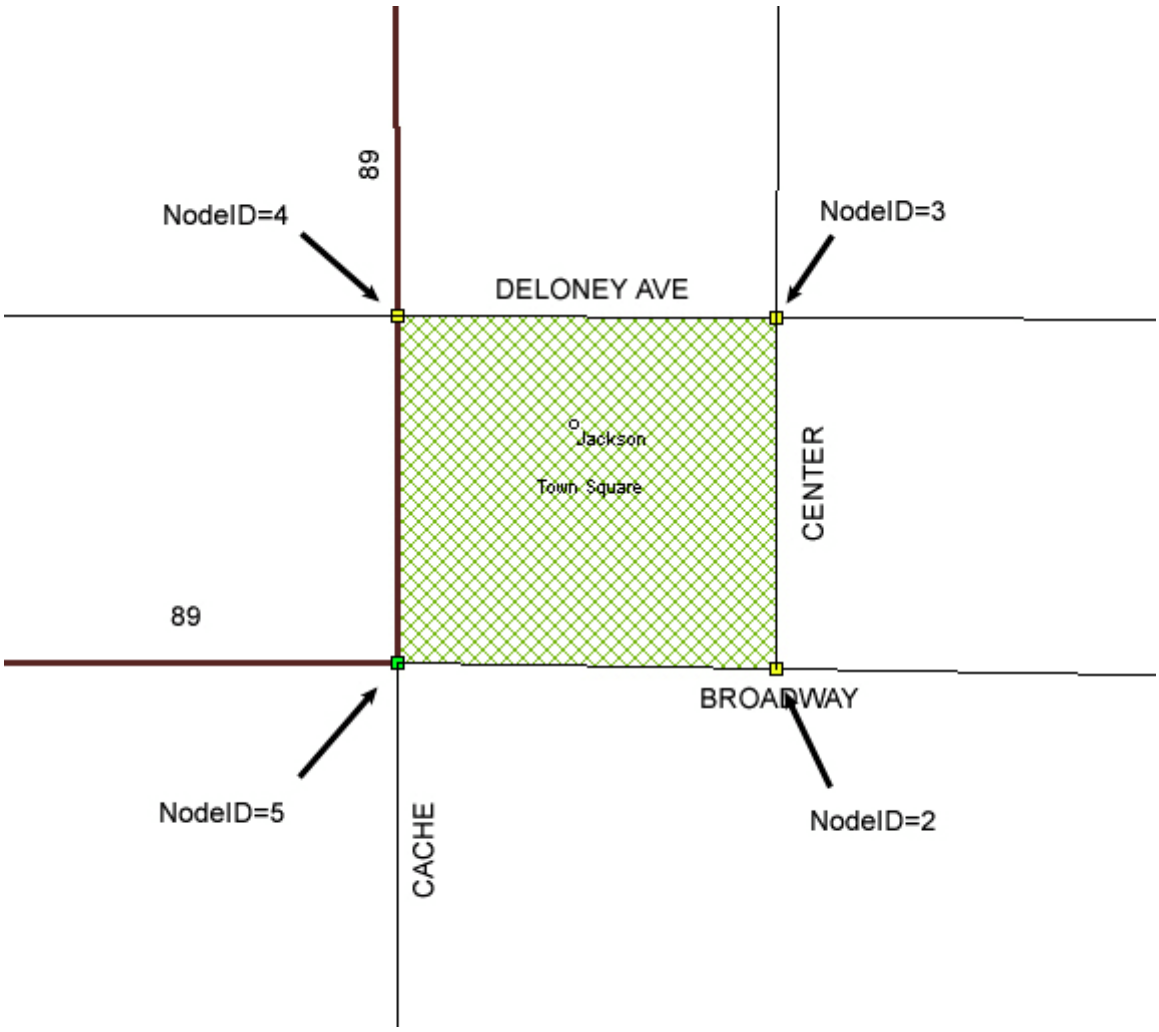
Routable maps have additional data associated with the road object declarations, indicating the speed limit, the class of road, and specific restrictions such as toll or one-way.

Currently, GM8 does not have the ability to edit all of these routing attributes, so we will use another program to help us with generating the routing information. GM8 is still a very important part of creating routable maps, as you will see.

How Routing Works

Each routable road has associated information describing its routing characteristics. Each intersection between routable roads is called a Routing Node. Calculating a route in the GPSr is simply a matter of looking at where you are, where you want to go, and evaluating the available paths to generate the best route within your defined road grid, according to your preferences.

This map fragment, grabbed from a GPSMapEdit screen, has four routable intersections indicated by the small rectangles at the intersections. The yellow rectangles (NW, NE and SE) indicate two roads meeting at the intersection, and the green rectangle (SW) indicates three roads meeting at the intersection.



Let's explore the Polish format source code underlying this map.

All of these routing attributes have been added by GPSMapEdit; we're simply going to take a look to understand what's going on inside the data file when routing information is present. I've reformatted the coordinates to make it easier to see the relationships between vertices used as NodeIDs, and added comments to the side explaining the meaning of the `Nodx=` declarations.

```
[RGN40]
Type=0x2
Label=89
Levels=2
RoadID=1
Data0=
(43.474510,-110.781707), ; <- 0th node
(43.474975,-110.777572),
(43.475101,-110.776176),
(43.475505,-110.774711),
(43.477554,-110.771392),
```

```
(43.478736,-110.769744),
(43.479419,-110.768470),
(43.479576,-110.768066),
(43.479671,-110.767440),
(43.479671,-110.762313), ; <-9th node
(43.480369,-110.762313), ; <-10th node
(43.488754,-110.762374),
(43.490890,-110.761878),
(43.495021,-110.760543)
Nod1=9,5,0      ; <- 9th node connected to NodeID 5 (Broadway & Cache)
Nod2=10,4,0     ; <- 10th node connected to NodeID 4 (Deloney)
[END-RGN40]

[RGN40]
Type=0x6
Label=BROADWAY
Levels=1
RoadID=2
Data0=
(43.479671,-110.762313), ; <- 0th node
(43.479660,-110.761261), ; <- 1st node
(43.479606,-110.756889),
(43.479724,-110.754425),
(43.479724,-110.752273),
(43.479724,-110.749900),
(43.479763,-110.743186)
Nod1=0,5,0      ; <- 0th node connected to NodeID 5 (89 & Cache)
Nod2=1,2,0      ; <- 1st node of Broadway connected to NodeID 2
(Center)
 [END-RGN40]

[RGN40]
Type=0x6
Label=CACHE
Levels=1
RoadID=3
Data0=
(43.479671,-110.762313), ; <- 0th node
(43.471702,-110.762336)
Nod1=0,5,0      ; <- 0th node connected to NodeID 5 (89 & Broadway)
[END-RGN40]

[RGN40]
Type=0x6
Label=DELONEY AVE
Levels=1
RoadID=4
Data0=
(43.480381,-110.767250),
(43.480369,-110.762313), ; <- 1st node
(43.480365,-110.761261), ; <- 2nd node
(43.480342,-110.753021)
Nod1=1,4,0      ; <- 1st node connected to NodeID 4 (89)
Nod2=2,3,0      ; <- 2nd node connected to NodeID 3 (Center)
[END-RGN40]

[RGN40]
```

```
Type=0x6
Label=CENTER ST
Levels=1
RoadID=5
Data0=
(43.479660,-110.761261), ; <- 0th node
(43.480365,-110.761261), ; <- 1st node
(43.481895,-110.761245)
Nod1=0,2,0      ; <- 0th node connected to NodeID 2 (Broadway)
Nod2=1,3,0      ; <- 1st node connected to NodeID 3 (Deloney)
[END-RGN40]
```

First, an explanation of the `Nodx=` declaration.

```
Nodx= <This road's zero-based coordinate pair reference>, <NodeID>,
<External (1|0)>
```

The first number represents the offset of the referenced coordinate pair, counting from 0. The second number is the NodeID. Any other `Nodx=` declarations with the same NodeID in this position share an intersection with this `Nodx=` declaration. The third number says whether or node this node is external, meaning that other maps can link to this map at this node. Its value can be 1 or 0.

For an example, let's look at the `Nod2=` declaration for DELONEY AVE and the `Nod2=` declaration for CENTER ST. These two statements share a reference to NodeID 3, creating the routable intersection of these two streets.

DELONEY AVE	CENTER ST
Nod2=2,3,0	**Nod2=1,3,0**
(43.480365,-110.761261)	(43.480365,-110.761261)

The second routing node from DELONEY AVE (**Nod2=2,3,0)** tells us that the second coordinate pair of this road (counting from 0) is a member of the NodeID 3 intersection, and that this is not an external routing node.

The second routing node from CENTER ST (**Nod2=1,3,0)** tells us that the first coordinate pair of this road (counting from 0) is a member of the NodeID 3 intersection, and that this is not an external routing node.

Notice that the referenced coordinate pairs are identical, as they must be for routing to work. GPSMapEdit will not create a routable node unless there are matching vertices at the intersection of the two roads.

Now let's look at the intersection defined by NodeID 5, with three intersecting roads.

89	BROADWAY	CACHE
Nod1=9,5,0	**Nod1=0,5,0**	**Nod1=0,5,0**
(43.479671,-110.762313)	(43.479671,-110.762313)	(43.479671,-110.762313)

Again, note that the coordinate pairs for this intersection are identical.

At compile time, cGPSMapper looks through the .MP file for all `Nodx=` statements, grouping those with matching Node IDs, and creating the routable road data structure that your GPSr scans when calculating a route.

***Understanding* `RouteParam=`**
In addition to declaring the routable road intersections, we also need to declare attributes for the individual roads describing the Speed Limit, and Road Class characteristics. In Polish format, this is done with a `RouteParam=` declaration:
`RouteParam``=<Speed>,<Road Class>,<restrictions…>`

Here is a table showing the possible values of `RouteParam=` for Speed and Road Class, the two most important parameters:

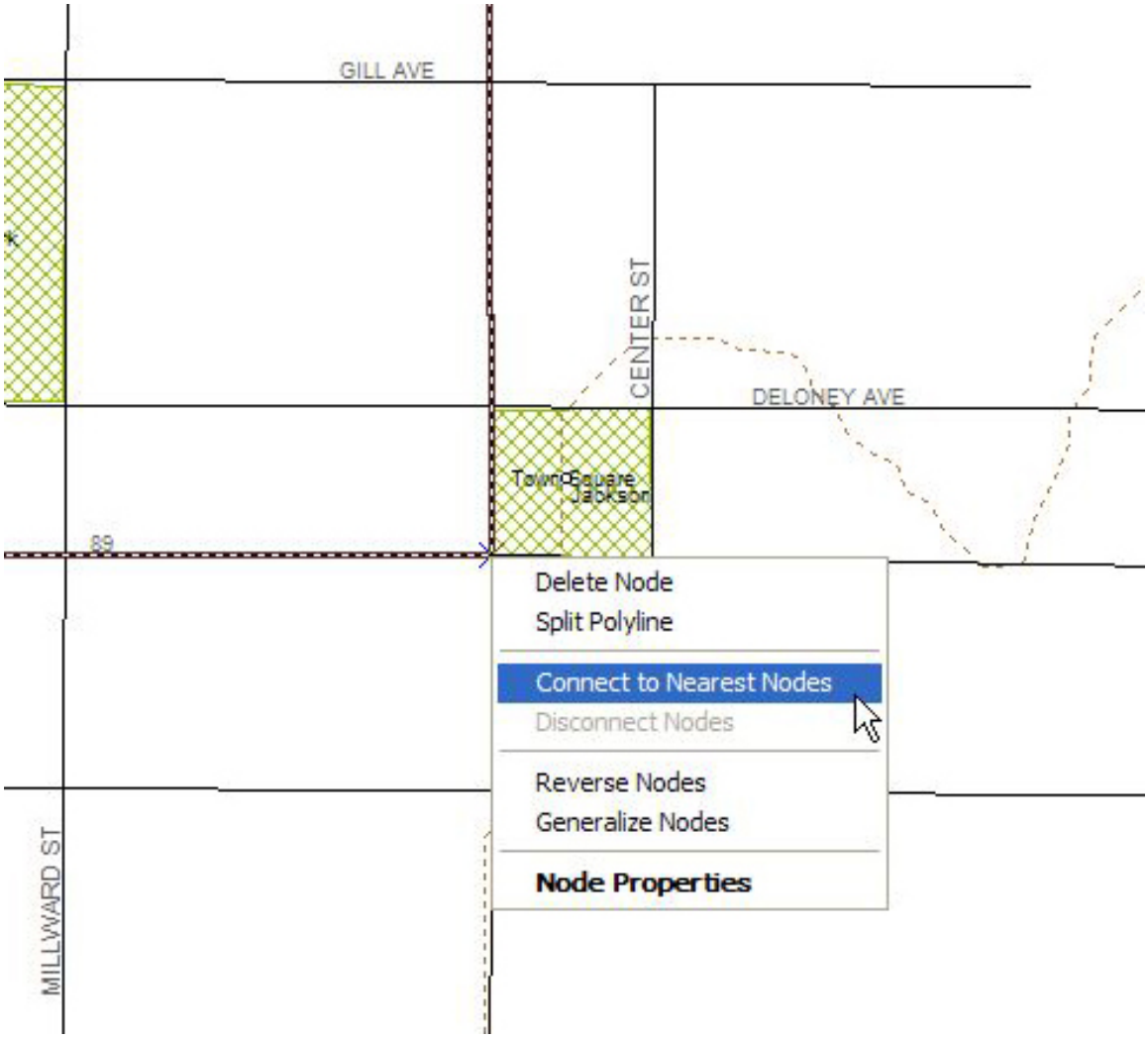
Speed			**Road Class**	
Parameter	MPH	KPH	Parameter	Road Types
0	3	5	0	Street/Alley/Unpaved
1	15	20	1	Collector/Roundabout
2	25	40	2	Arterial/Minor highway
3	35	60	3	Principal highway
4	50	80	4	Major highway/ramp
5	60	90		
6	70	110		
7				

The Speed parameter is used when calculating the fastest route and projected arrival time. The Road Class attributed is used to select the path of least resistance through an area. Higher numbers mean more desirable routing.

Creating Routing Nodes
Now we're going to create the data that we've been studying here.

Launch GPSMapEdit, and open Jackson WY.MP. Press the ***Z*** key to enter Zoom mode, then drag a box around the center of town.

Press the ***M*** key to enter the Edit Nodes mode, then click Highway 89 so that it is selected. Right-click at the intersection of 89, Broadway and Cache, then select ***Connect to Nearest Nodes***.



This creates a routing node at the intersection of 89, Broadway and Cache. The resulting node should be green, indicating a 3-street intersection. When we created this intersection in GM8, it was an origin point for each of the roads, so there are already vertices at this location to connect. GPSMapEdit shows vertices as 'X's. Note that there are no vertices at the intersection of 89 and DELONEY, just north of our 3-street intersection. GPSMapEdit requires vertices in both intersecting roads at any location intended to become a routing node.

Right-click at the intersection of 89 and DELONEY, and note that there is no option to ***Connect to Nearest Nodes***. We need to add vertices at the intersection of these roads. With Highway 89 still selected, position the cursor at the intersection of 89 and DELONEY. Right-click, then select ***Add Node Here***. Click on DELONEY, position the cursor at the same intersection, then right-click and select ***Add Node Here***. Depending upon how accurate you were with your clicking, you may or may not have a right-click option to ***Connect to Nearest Nodes*** at this intersection. I usually have to drag one of the vertices towards the other until it lights up red, indicating that they're aligned before I can use ***Connect to Nearest Nodes***.

As you can see, this process will get tedious very quickly! Fortunately, GM8 can help us out.

Close GPSMapEdit without saving any of your edits, and return to GM8. Use the Overlay Control Center to hide all but the User Created Features overlay. Press ***Alt+D*** to enter Edit Mode, press ***Home*** to show your entire map.

Click ***Tools | Configure…*** (or the Configure icon  in the toolbar) then click the Vector Display tab. In the Select From section, select only Lines, then click OK. Drag over your entire map so that all the roads are selected. Press ***Shift+V*** to toggle vertex display so that vertices are visible. Right-click anywhere, then select ***Insert Vertices at Intersections of Selected Features***. Note that GM8 has created coincident vertices at all road intersections for us.

Press ***Esc*** to unselect the roads. Let's use this opportunity to do a little clean up on some Areas.

Click the Configure icon in the tool bar to open the Configuration dialog, then click the Vector Display tab. Select only Areas, then exit the Configuration dialog.

Press ***Alt+Z*** to enter Zoom Mode, then drag over Miller Park. Press ***Alt+D*** to enter Edit Mode, then click near a corner of the park area such that the vertex is highlighted. Right-click and select ***Move Selected Vertex***. Click and drag the vertex so that it snaps to the road intersection. Repeat for all four corners, then repeat for the Fairgrounds and the Town Square. When you're finished tidying up, open the Vector Display tab in the Configuration dialog to restore the checkboxes for Areas, Lines and Points. Press ***Ctrl+S*** to save your workspace.

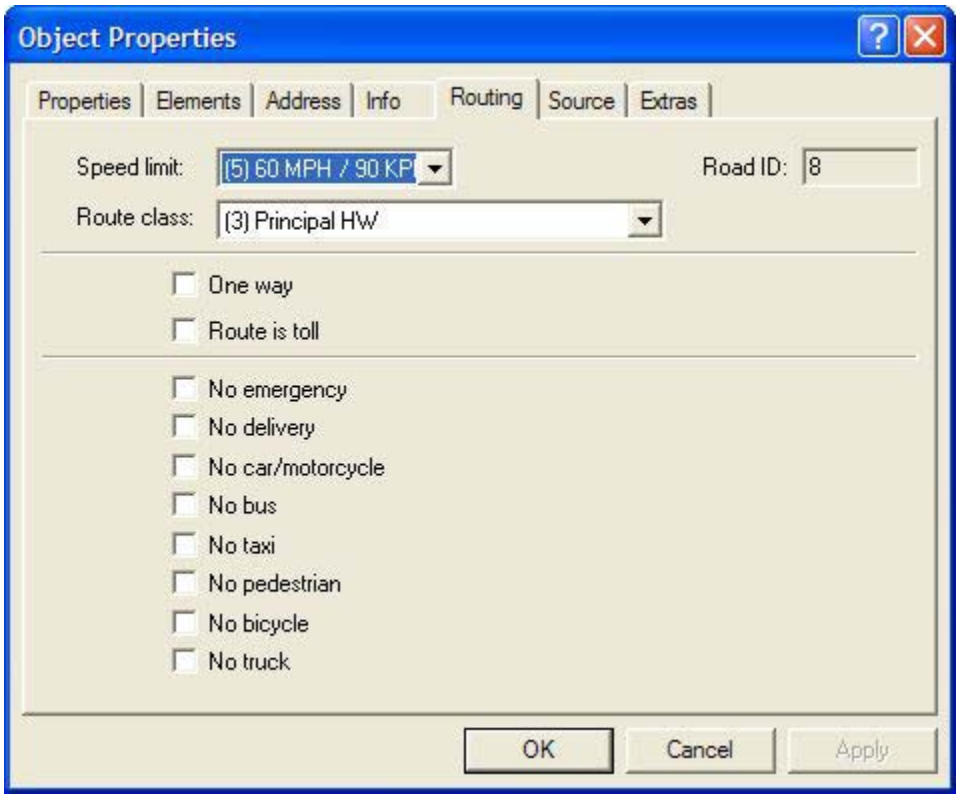
Export your Polish format file, overwriting the previous copy. Open this file in GPSMapEdit.

Press the ***M*** key to enter Edit Nodes mode, and click on Highway 89. Note that there is now a vertex at the DELONEY intersection. Click on DELONEY, and notice an identical vertex at that intersection. This map is ready for routing nodes.

In the GPSMapEdit menu, click ***Tools | Generate Routing Nodes | At Coinciding Points of Polylines***. This automatically creates routing nodes at all intersections. The blue nodes are created at the ends of roads - don't worry about them. Check your map to make sure that all intersections are at least yellow, and not blue. If you see blue at an intersection, it probably means that the one of the roads terminates just shy of the intersection. Press ***M*** to Edit Nodes, then click and drag the vertex to align it with the other, then right-click to ***Connect Nodes***.

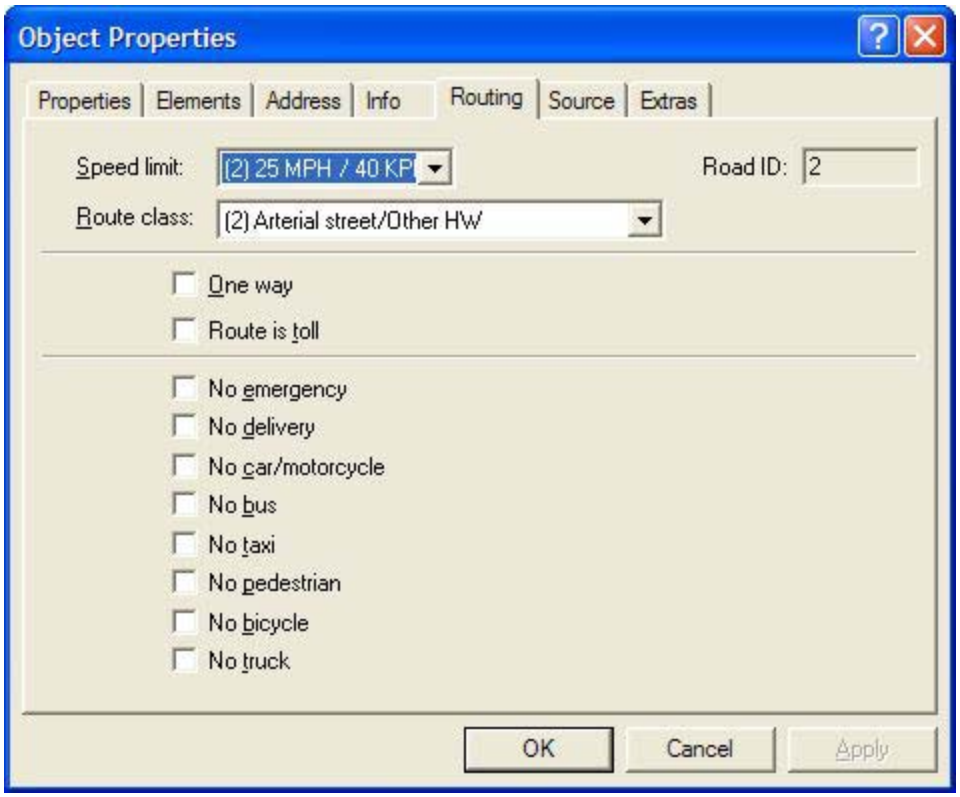
Rectangle Color	Meaning
Blue	Routing node, no intersection
Yellow	Intersection of 2 roads
Green	Intersection of 3 roads
Purple	Intersection of 4 or more roads
Red	Routing node with errors
Red border	Node has turn restrictions
Large node	Node is external

Now let's take a look at how GPSMapEdit assigned Speed and Route class attributes. In GPSMapEdit, press the ***S*** key to Select Objects, then double-click on Highway 89 to bring up the Object Properties dialog, then click the Routing tab.



GPSMapEdit assigned a default speed of 60mph, which is too fast for this segment through town. In the Speed limit dropdown, select (3) 35 MPH / 60 KPH. Click OK to accept the parameters and dismiss the Object Properties dialog.

Now double-click BROADWAY, and adjust its parameters to Speed limit (2) and Road class (2).



Click OK to accept the parameter changes and dismiss the dialog. Apply the same settings for CACHE. We now have the most important streets in town designated as Principal HW or Arterial. Let's take a look at the other streets to see what `RouteParam=` attributes have been assigned.

Double-click Pearl, and note that it has a Speed limit of (3) and a Route class of (0). That's probably too fast for a city street. All of the other city streets will be the same. We'll use GM8's powerful Search feature to edit all of the city street attributes simultaneously.

Save the file in GPSMapEdit by pressing ***Ctrl+S***, then exit.

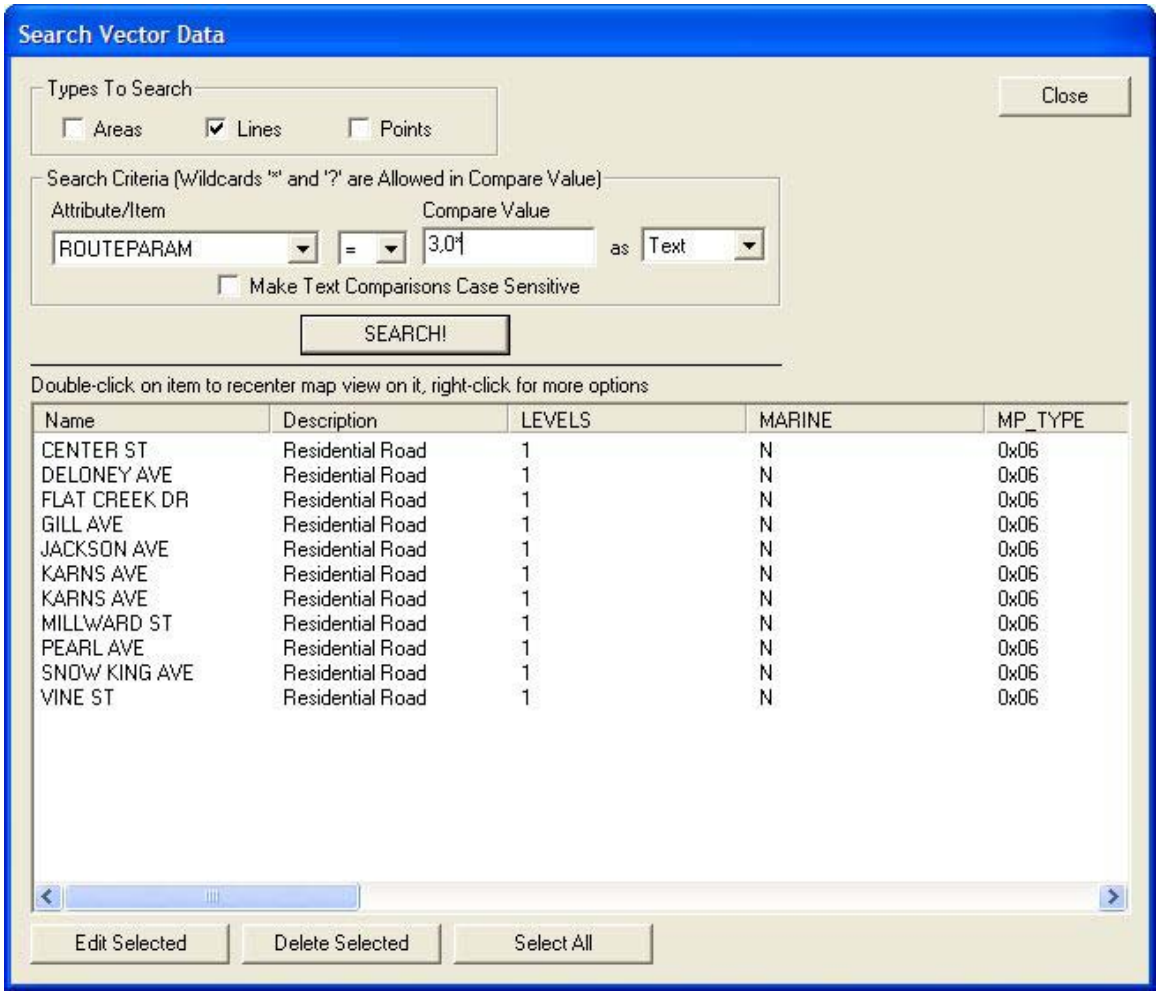
Now we need reload our Polish format file that's been edited in GPSMapEdit. Our workspace already has a representation of the roads in our USER CREATED FEATURES overlay, but we now want to work with our edited version. First, save your current JACKSON WY.GMW workspace just in case you want to come back to your original data.

Open the Overlay Control Center, select USER CREATED FEATURES, then ***Close Overlay***.

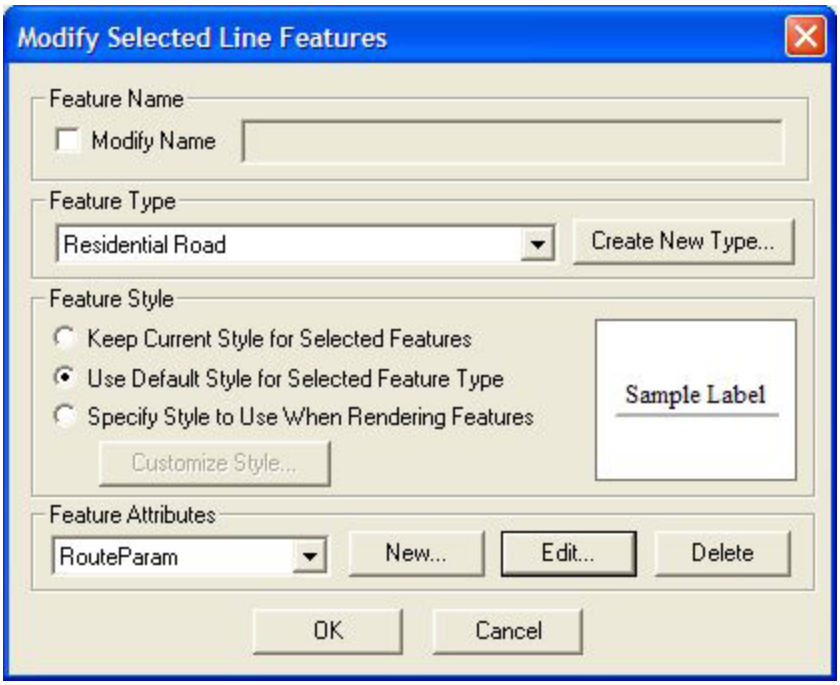
Press ***Ctrl+O*** to Open File, and open JACKSON WY.MP – the updated vector feature file that was just saved by GPSMapEdit. Click ***File | Save Workspace As …*** , and save this workspace as JACKSON WY IMPORT.GMW. This version of the workspace will use the imported .MP file with our edits.

Press ***Home*** so you can see the entire project. We're now going to use the Search feature to find all roads whose `RouteParam=` attributes begin with 3,0 – in other words, those roads for which GPSMapEdit set a speed limit of 35.

In the menu bar, click ***Search | Search by Attributes, Name, and Description***.
In Types to Search, click 'Lines'. In Search Criteria – Attribute/Item, select ROUTEPARAM from the drop-down list. Select '=' from the comparison type list, and '3,0*' as the Compare value. This will find all lines whose RouteParam= attribute begins with 3,0.



This is exactly the group of roads we're looking for. Click 'Select All', then 'Edit Selected'.



In the Modify Selected Line Feature dialog, select RouteParam from the Feature Attributes drop-down list, then click Edit. Change the initial 3 to a 2, representing a 25mph speed limit for city streets. Close all open dialogs and return to the GM8 main screen.

> ***Note**: Beginning with version 1.0.32.0 of GPSMapEdit, it is possible to simultaneously change the Speed and Route class parameters for a group of selected roads. This section of the tutorial was designed to introduce the powerful Search/Modify features of GM8.*

Now we'll export our modified road database plus the topo contours. Open the Overlay Control Center, display the GENERATED CONTOURS overlay, then close the Overlay Control Center.

Export the visible vector overlays as JACKSON WY.MP. Note that we have now merged our two vector overlays into a single overlay.

Compile and download your map to your GPSr.

This concludes the introductory tutorial sequence.

Tutorial 4a – The Development Cycle

To continue developing your map, you need to be aware of a few key concepts to keep everything in order.

Only vector data can be compiled and downloaded into a GPSr. In these tutorials, we combined the digitized features that you created manually with GENERATED CONTOURS. If you want to remove or exclude vector features from your export, you can easily do so by using the filter functions of GM8. You will find that GM8 is very flexible in its ability to manipulate your database.

If you want to edit your .MP Polish format source files outside of GM8, you need to follow a simple sequence to keep things in order.

When you are ready to work on your source file outside of GM8, export it as a .MP Polish format file.

Edit the file as needed in your external tool, saving it again in .MP format.

When you are ready to re-import the modified file to GM8, open your saved workspace, and Close the older version of your data in the Overlay Control Center. This removes the old version from your workspace. Use the ***Open Data File(s)*** command to re-import your new file, reintegrating it into your workspace.

Have fun!

Additional Online Resources

Global Mapper Tech Support Group

http://tech.groups.yahoo.com/group/global_mapper/

cGPSMapper Tech Support Group

http://tech.groups.yahoo.com/group/map_authors/

GPS Passion Forum

http://www.gpspassion.com/forumsen/default.asp