Brush patterns in MapInfo

Jacques Paris

jacques@paris-pc-gis.com

May 2001
Revised may 20

MI uses bitmaps stored in a DLL to create fill patterns by tiling them over the surface of the objects. Once they have been identified and that their nature and how they can be accessed is well understood, it is easy to modify them and to add to them. This paper deals with the basic rules to respect and shows how to use some tools to build costume pattern palettes.

MI emploie des bitmaps contenus dans une DLL pour créer des motifs de remplissage en faisant une mosaïque de ces images sur la surface des objets. Une fois l'identification de ces bitmaps faite et leur nature et la façon d'y accéder bien comprises, il devient facile de les modifier et d'en ajouter. Cet article montre les règles fondamentales à respecter et comment utiliser certains outils pour construire des palettes de motifs sur mesure.

Generation of object fills

MI fills the interior space of closed objects by tiling over the surface of the object a simple 8x8 bitmap, trimming the excess beyond the boundaries. The tiling of the basic bitmap will create the “brush patterns” that are offered to the user.

The result is scale independent in the sense that a change of zoom will not modify the look of the pattern. It is the way MI treats symbols and labels (fixed size) but not text objects (proportionate to zoom).

Basic Bitmaps

The bitmaps that are used as base for the fill patterns are stored in the MIRESnnn.DLL file (nnn stands for the version number, 650, 600, 550, …), in the section “Bitmap”. They have the “names” 1 to 61 and are related to the pattern numbers (those used in the Brush clause) by a simple rule preceded by exceptions.

It is worthwhile noting that 6 patterns are duplicated: patterns 3 through 8 are similar to ~~can be found again as~~ 22 (12+10), 26, 29, 34, 41, and 44. They are however independent of those bitmaps, being apparently “hard coded” [1]

All the basic bitmaps (fill01.bmp to fill61.bmp) are zipped in the MI_patterns.zip file as part of a subdirectory containing also the image (all_fills.bmp) of the pattern choice popup window part of the region style requester.

Brush Pattern	Bitmap#
02	1
03	12*
04	16*
05	19*
06	24*
07	31*
08	34*
12	2
13	3

from 2 to 71

brush pattern = bitmap# + 10

* similar to this bitmap in the original DLL

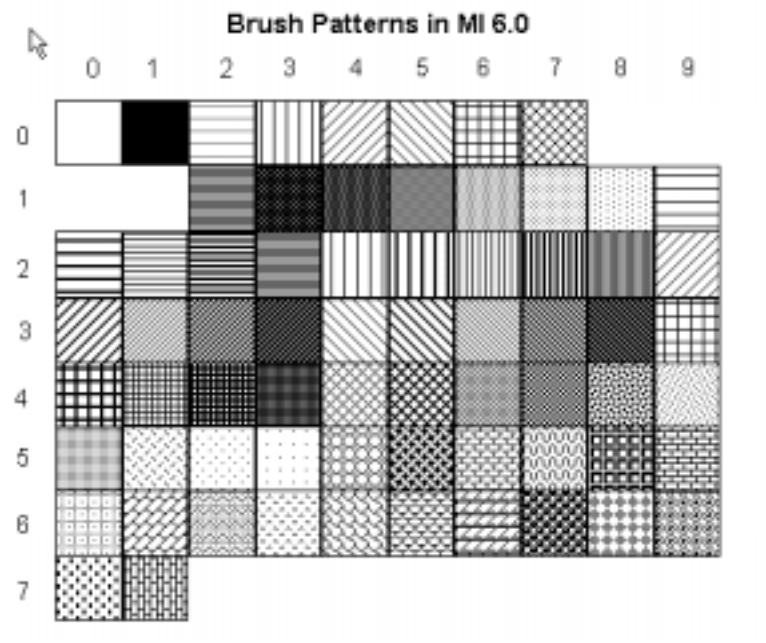
The bitmaps must be defined as MONOCHROME (or 2-bit color) and not many programs give that choice. If another format may be accepted “by” the DLL, the result may be a monochrome pattern (with the color used to fill in the bitmap) that cannot

[1] Correction of May 17, 2001

accept a background color (it disappears entirely) and remains black whatever the color assigned to the foreground.[2]

Brush patterns

The available brush patterns are presented in the following map. The corresponding MI table is part of the MI_patterns.zip file and can be put to different uses.



Graphic performance of "tiled" patterns

The graphic performance[3] of tiled patterns can be appreciated on the screen and in printed form. What can be acceptable for one may turn out useless on the other. The result is a conjunction of the technical characteristics of the hardware (monitor and printer) and of the program (how MI displays basic pixels, how it translates them in printer code).

We will not move into hardware considerations, there are too many variations possible to cover them, but as a way to appreciate the performance of its monitor, the user can

[2] Addition of May 19, 2001

[3] I understand graphic performance on two levels. One, the possibility to distinguish clearly between the patterns of a "legend", and two, the message attached to each pattern: symbolic meaning is generally connected to the traditions and practices of the particular "mapping" milieu.

simply display the pattern_map table presented above, and for his printer, to print the mapper.

Zooming in and out on a pattern gives also a good idea of what it could look in small or large surfaces.

Creating new bitmaps

Looking at existing patterns (the fillxx.bmp in the MI_patterns.zip file must be magnified at least 5 times to be clearly seen), one gets a quick appreciation of the talent required to develop new patterns. The constraints are many. There is for example the overall size itself that has some impact on:

- the "step" within the pattern: a single design is repeated every 8 pixels in both directions, smaller designs must be repeated in the 8x8 matrix and that leaves only the choice of 2x2 and 4x4 designs if geometric continuity of the pattern is essential;

- the " maximum separation" between designs: elements of a bitmap (e.g. pixels in a simple straight line) cannot be separated by more than 7 pixels from the corresponding elements of the neighboring tiles; thus vertical or horizontal one-pixel lines separated at the most by 7 white pixels, two-pixel lines by 6 white…

- all the elements of a pattern must be defined in a 8x8 matrix which is sometimes difficult to figure out for "diagonal" designs where continuity of the pattern is obtained by the right positioning of the opposite edge pixels

One can also reflect on the size of a pixel, essentially the visual impact one pixel as defined in the pattern and its "translation" both on screen and on paper. A single black pixel in a sea of white is more "visible" than a single white on black.

Any simple bitmap editor would be good enough to create a pattern.BMP. Some may have more tools to facilitate the work, but essentially we are looking at a software that will be able to generate bitmaps respecting some limited but fundamental requirements.

Beside the size and the format, the BMP must be saved with a "monochrome" palette. BMPs saved without that feature may be refused by the resource editor as having a "bad format" for Windows.

On a very practical level, when designing a new bitmap, it is generally difficult to imagine the result when tiled. It would be very useful to rely on a tool permitting to work on a blown up 8x8 matrix and to see in a separate window the result from the tiling of that matrix.

I have found one that has that capacity, but the search should be continue to find a "better" tool, if it was possible. "Tile Studio" by Mike Wiering, Katholieke Universiteit Nijmegen, available as freeware at http://www.cs.kun.nl/is/ts/ . It is a "overkill" tool for our

purpose but it does what we need quite simply and the interactivity tile > pattern is quite effective.

Some steps have to be taken carefully to avoid unnecessary operations later on. The purpose of the "instructions" given at the end of the document is to demonstrate how easy it can be and for new users, to avoid having to uncover by themselves the right operations.

One requirement remains paramount: each time a pattern has been modified, or once a new one has been created, it must be saved in a monochrome bmp format. If the edit program does not offer that possibility (and Tile Studio version 1.3 does not), another program is required. It could as simple as Microsoft Paint that comes with Windows because it is just a question to open the bitmap and save it in the appropriate format.[4]

Modifying the existing pattern palette

If someone wants to bring some changes to the patterns of the default palette, he can do so in a very limited way and respecting some strict rules. But before starting, let us stress that the solution that is offered here (and I do not see any other for now) implies that the MIRESnnn.DLL must be rebuilt each time MapInfo comes up with a new of version.

Another constraining consideration is of importance if an application (meaning a WOR) is going to be open on an other installation: the modified version of the DLL must also be present on that installation. That means the risk of having version conflicts (each MI version has its MIRES version) and troubles with the network administrator as the dll is part of the "central" installation.[5]

Using different pattern palettes

The first considerations are purely technical and deal with the general MapInfo setup. MapInfo expect to find the MIRESnnn.DLL in its directory whenever it calls for it. There cannot thus be a "change on the fly" of the pattern palette. The entire DLL must be "changed", that is, a different version of the DLL must be "active". ~~I am pretty sure that the~~[6] The switch cannot be done while MapInfo is running.

To do that, one must make a copy of the original DLL and save it under a different name (let us say MIRESnnn_ORI.DLL). Then one makes another copy and saves it under a specific name (let us say MIRESnnn_ONE.DLL) and brings the needed changes to the copy before saving it. A safe way to make the switch of the active DLL is to delete the MIRESnnn.DLL, make a copy of the file to be used and rename that copy without its _XXX part.

[4] Addition of May 19, 2001
[5] Addition of May 18, 2001
[6] Correction of May 17, 2001

Constraints on the bitmaps

The second batch of limitations comes from the basic bitmaps themselves. First their number is exactly 61~~, for the 67 pattern ids available~~. New patterns mean loosing old ones, except for the 6 that have been identified as having "hard coded" equivalents (12, 16, 19, 24, 31, 34).[7]

Second, their "names" (the pattern number ids) must remain exactly what they are. Finally, the images must measure exactly 8x8 pixels and be in black and white (monochrome bmp).

Modifying the DLL: Relying on outside resources

Once the bitmaps have been created, the DLL can be modified by editing its contents. One will use some "resource editor" program, such as "Resource Hacker" which has the advantages of doing this job very simply and cleanly, and of being free. The sequence of operations are detailed at the end of this document.

Resource Hacker, by Angus Johnson http://rpi.net.au/~ajohnson/resourcehacker/

Adding new patterns to the MI palette

The changes can also be additions. In MIRESnnn.DLL, there is a sizeable gap in the numbers representing the bitmap names between 61 (the first 61 bitmaps are fills) and 900 (the first own of images used by MI in its menus). MI will reject a style definition (such as MakeBrush) if the pattern value is higher than 255, which means that bitmap names are acceptable up to 245. If MI does not decide to use these numbers for other purposes, it can thus support 3 times as many patterns as it offers presently.

If bitmaps are added with consecutive numbers starting at 62, the corresponding patterns will be displayed in the region style requester and be readily available (I have not checked if there was a limit to the number of displayable patterns).

If there is a gap in the numbering, the bitmaps found after the gap will not be displayed in the requester. However, the patterns corresponding to all the bitmaps can be used in any Brush definition by adding 10 to the bitmap "name" even if they are not displayed, as long as they remain within the acceptable range (below 256 for patterns, 246 for bitmap names).

The procedure to follow for adding patterns when using Resource I described at the end of the document.

[7] Id.

Collecting new patterns

I have added in the MI_Patterns.zip file some bitmaps I designed very quickly essentially for demonstration (in the directory nu_bitmaps fill_jp_01 and up); they can be used to test the procedure for replacing old patterns or adding new ones and to experiment with their use in MapInfo. They may become the seed to a large and interesting collection.

Anyone who has developed new patterns and is willing to share them can send me either a complete MIRESnnn.DLL or even better only the basic BMPs. You can be assured that I will “distribute” them with full credits and documentation.

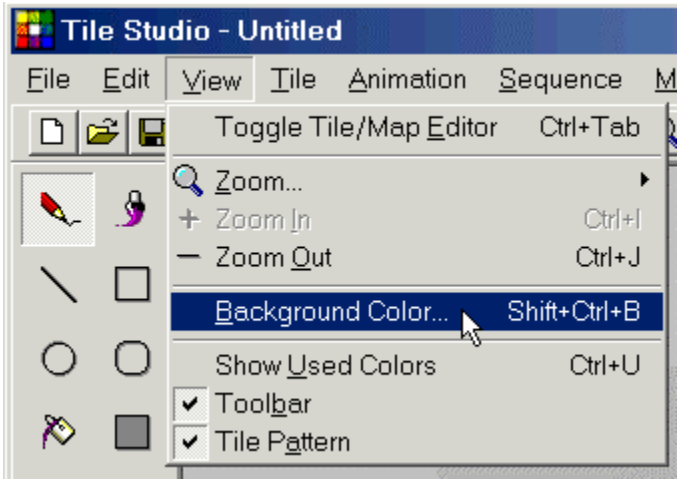
Making tiles with “Tile Studio”

Creating a new tile

An existing bitmap may serve as a basis for a new one. Changing few pixels may sometimes give a different impact to the new pattern or add a variation in a set of patterns.

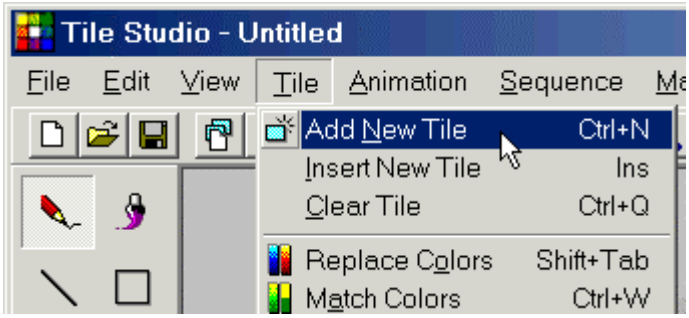
One must make sure that the background color is properly set to WHITE. This is required for a monochrome image; it will be the color that MI will treat as “background”, in fact transparent.

menu View | Background Color



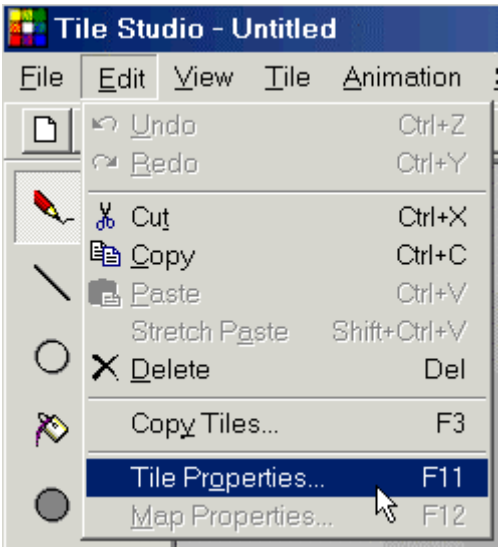
Clicking on “Background Color” opens a standard color requester. Choose WHITE. That operation has to be done once after T.S. is open.

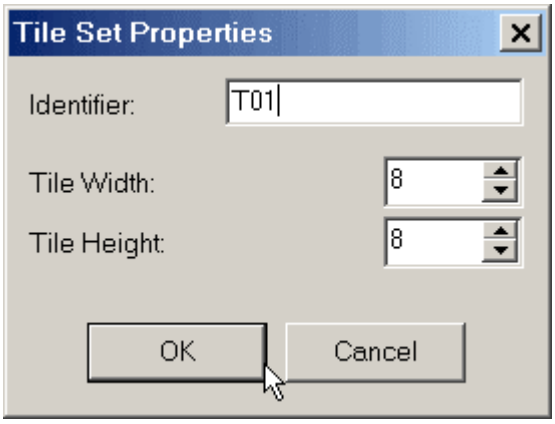
menu Tile | Add New Tile



The first time one adds a tile, it will open with default values. These must be set properly with

Menu Edit | Tile Properties





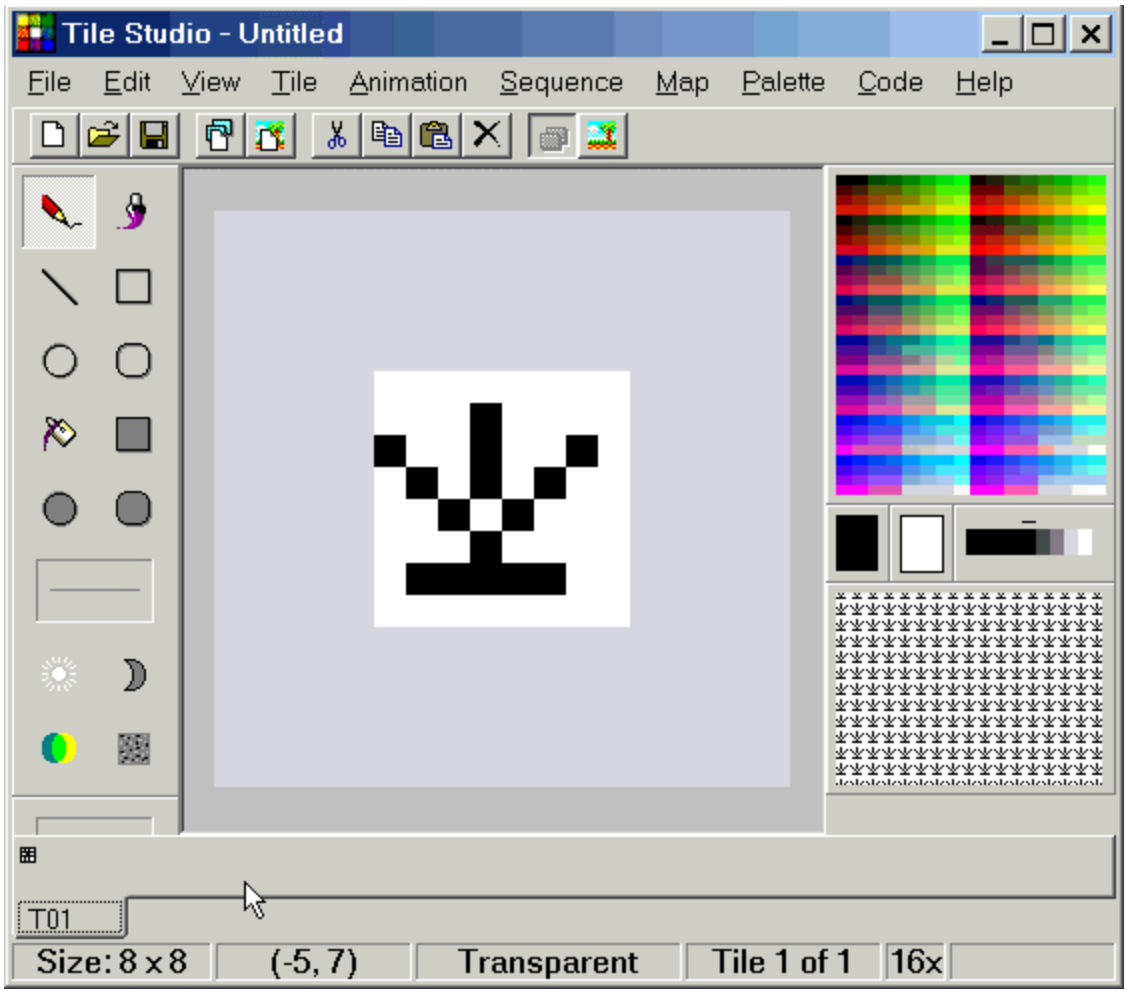
Choose the identifier of your liking, it will be used for the T.S. session only.
Make sure that the width and height are set at 8.

Any new tile will have those properties.

The T.S. window has been reduced to its minimum since without squishing the work area. The zoomed in view and workspace of the bitmap is the central square area. The pattern viewing window at the right bottom corner of the T.S. window has not been reduced in size.

The displayed pattern shows that if one wants a symmetrical pattern based on a central one-pixel line, that pattern cannot “cover” the entire 8x8 square. It will measure at the most 7 pixels in the other direction than the symmetry axis (7 wide for symmetry relative to the vertical axis).

For most applications, one needs only to use the pencil tool and the two largest boxes black and white to switch between colors.

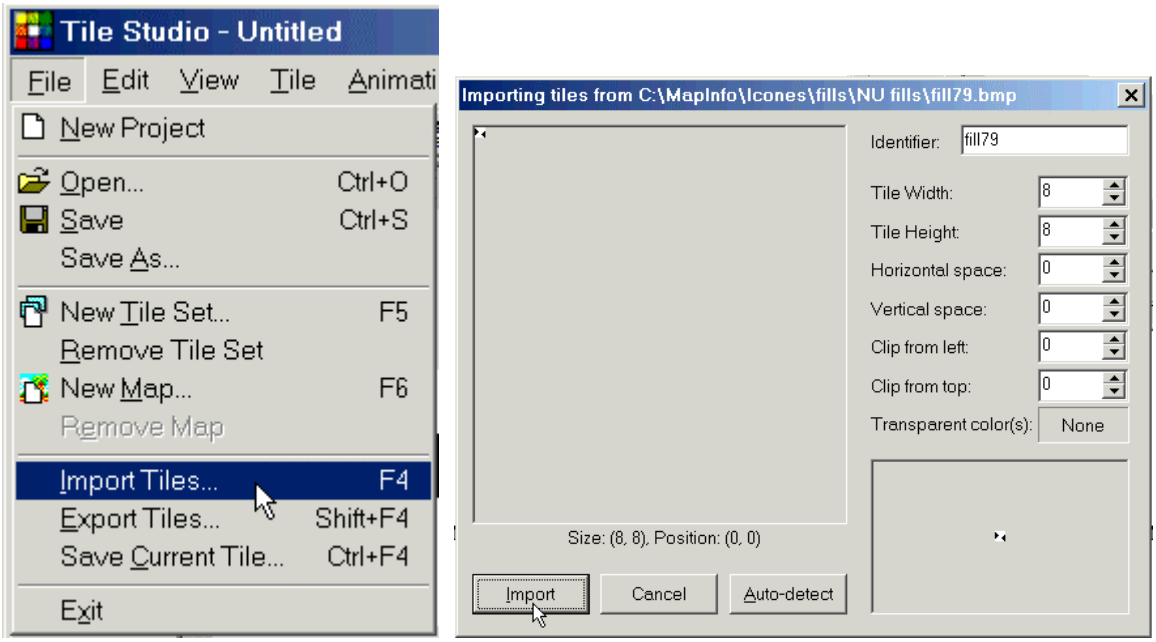


n.b. Several bitmaps can be loaded at the same time. Switching between them is simply done by clicking on the desired identifier tag, but if several images are defined with the same identifier (adding new tiles without changing the identifier in the Tile Properties requester), choose the image among those displayed in the bar below the workspace (in the image above, there is only one above the T01 tag)

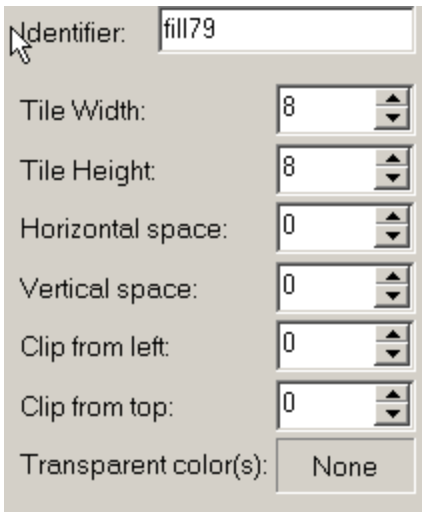
Importing a bmp fill

Starting with an existing bitmap requires importing that file.

menu File | Import Tiles



Clicking on “Import Tiles” opens the requester to the left above (reduced size image, a full size section below)

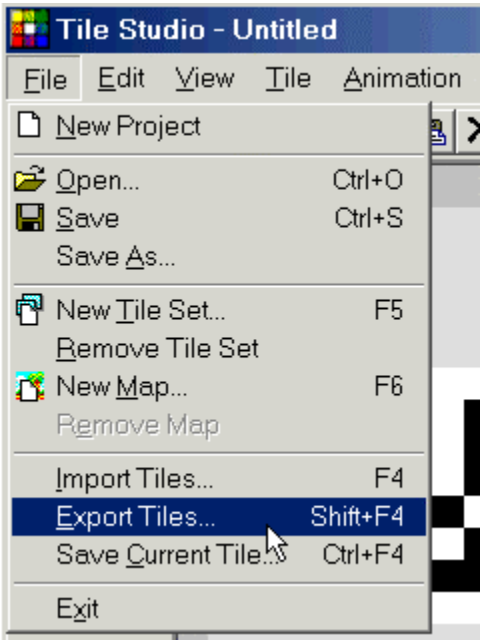


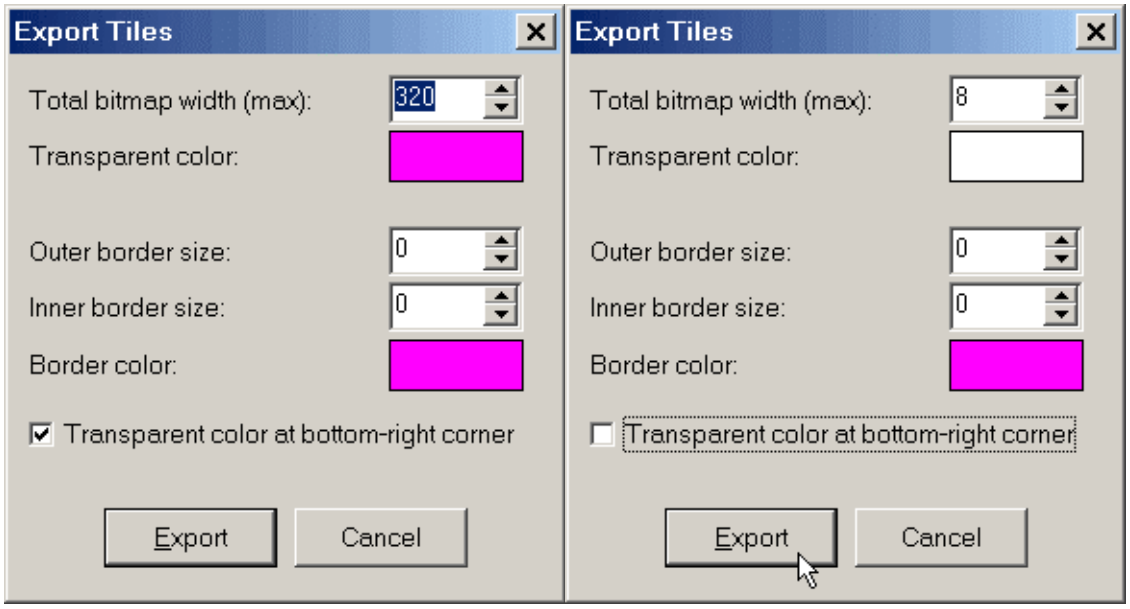
The identifier is automatically assigned the name of the bmp file. The user must enter the right width and height values (8), all others being 0, and make sure that transparent color is “None” (a right click on the mouse in that box will do)

Saving a bmp fill

We are interested in BMP files and not files in Tile Studio format. The Save or Save AS command will not do the trick.

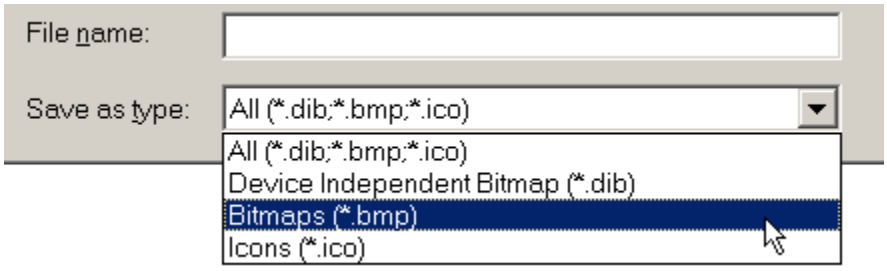
menu File | Export Tiles





The first time that operation is carried out, the values must be changed in 3 places: Total bitmap width (8), transparent color set to “white”, and unchecked Transparent color… at the bottom.

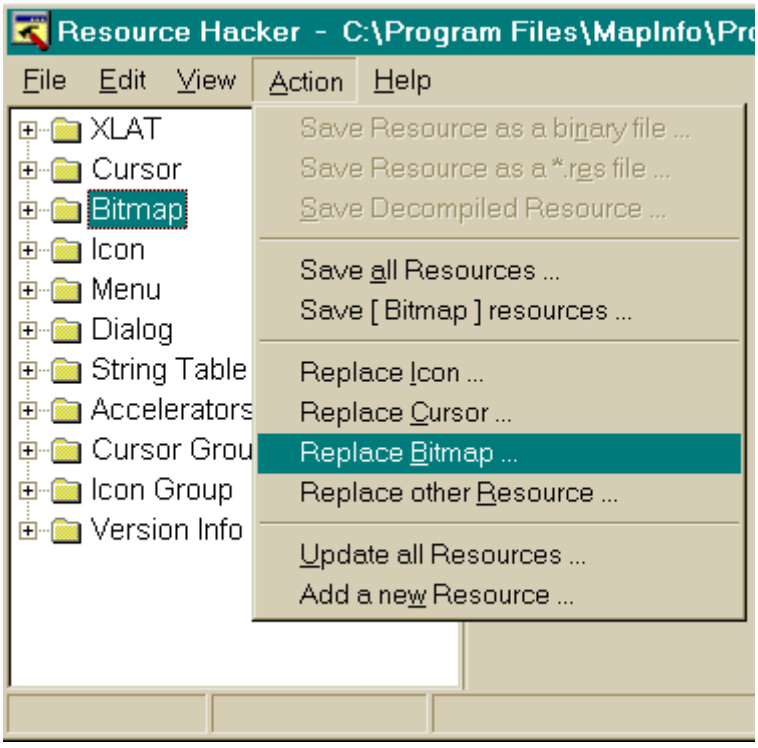
When the Save File requester opens, select the bmp format and give a name to your image.



Modifying the existing bitmaps with “Resource Hacker”

launch Resource Hacker

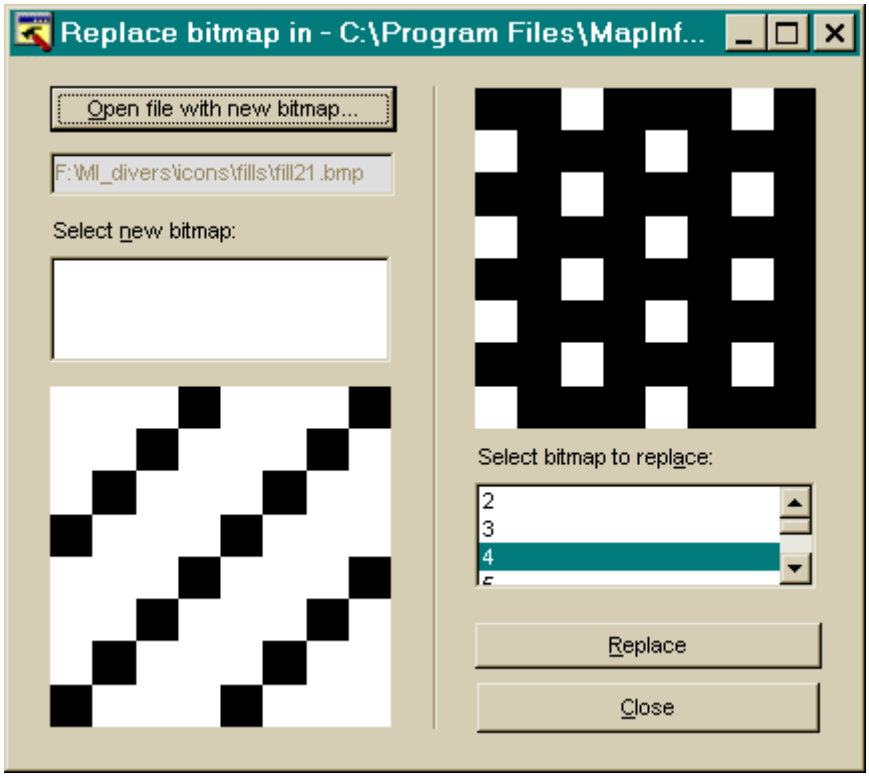
menu File | Open … select MIRESnnn_XXX.DLL in the MapInfo directory



menu Action | Replace Bitmap

Clicking on Replace Bitmap opens the following requester.

requester Replace Bitmap in …



Open file with the new bitmap locate and select the new bitmap
the new image is displayed in the bottom left part of the requester

Select bitmap to replace select the existing bitmap (name) number in the list
the existing image is displayed in the top right part of the requester

Replace or close

Repeat procedure for each bitmap to be changed

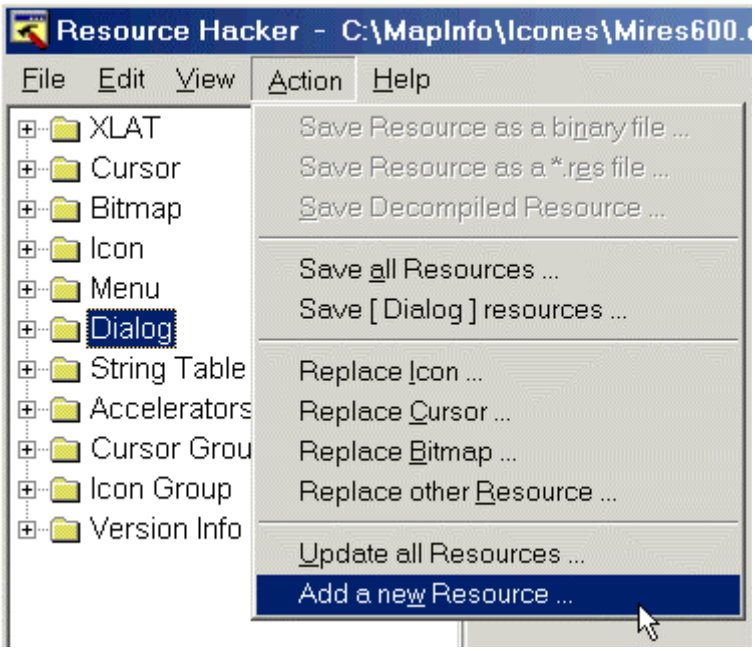
When all the changes are done

menu File | Save

Adding to the existing bitmaps with “Resource Hacker”

launch Resource Hacker

menu File | Open … select MIRESnnn_XXX.DLL in the MapInfo directory



menu Action | Add a new Resource

Clicking on “Add a new Resource” opens the following requester.

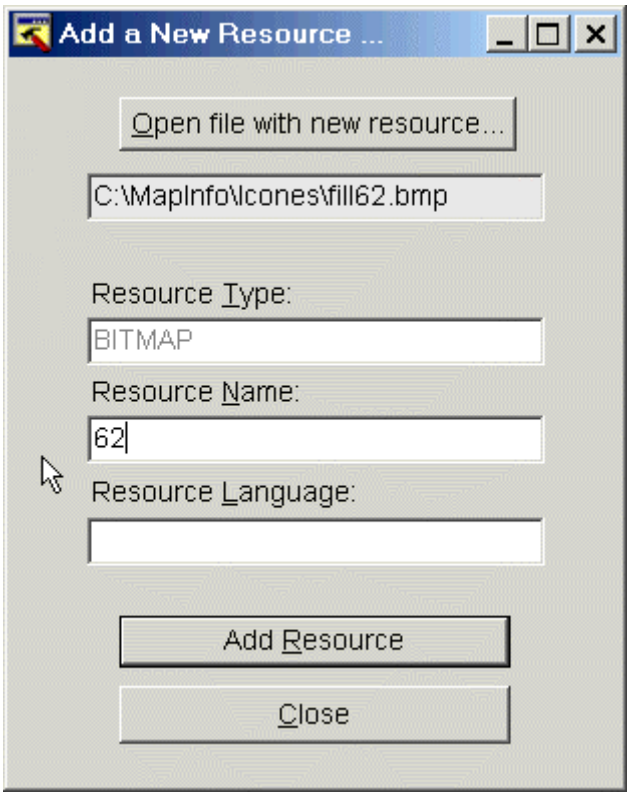
requester “Add a new resource”

1- **Open file with new resource**…
Clicking on it opens a file requester. Select the appropriate BMP.

2- **The resource type is immediately recognized**.

3- **Enter the Resource Name as a number**.

4- Ignore “Resource language”



5- **Click on Add Resource** (or Close)

When all the changes are done

menu File | Save