Handling MapInfo TEXT OBJECTS

for the Windows 32-bit platform only

=+=+=+=+=+=+=+=+=+=+=+=

This document is in no way an “official” summary on the subject.
It is the result of my experimentations under MI/MB 6.5
and several questions remain still unanswered.

Jacques PARIS

jacques@paris-pc-gis.com

29 April 2003

This investigation will hopefully leave us with a better understanding of what are text objects in MapInfo for the essential purpose of becoming better at handling them. There are some limited differences between text objects in a table (equivalent of geographic objects) and in a layout window (purely graphic objects); they will be noted where appropriate, but otherwise, what is said using the context of tables is applicable to layouts.

MI definition of a text object

The elements contributing to the definition of a text object fall into two main groups: container and contents. The container is essentially the spatial information that specifies its location, its size and the aspect of the text (essentially the angle it forms with the horizontal). The contents are the string itself and the font to be used for its display together with some internal layout parameters (alignment, line spacing and line style for labeling).

There are very strong relationships between these two "groups". A change in a contents element may have an immediate effect on the spatial definitions; for example, increasing the line spacing of a multiple line text will result in an increase of the dimensions of the text box. Conversely, modifying some spatial element can affect the way a text object is displayed; a change in the height of the box will trigger a change in the size of the displayed string. But this interrelationship is not systematic which may create some problems, and there is some interdependence between elements of the same group.

We should first introduce two geometric definitions that are crucial for the understanding of these objects. In a text object, the geometry is defined in two different ways:

> the **"text box" (TB)** is the rectangle that holds the string; its height defines the point size of the string. It can be set at any angle while keeping its shape. It is the area used for selecting an object (clicking outside that area does not select it). The upper left corner is the anchor point (center around which the text can be rotated)
>
> the **"text bounding rectangle" (TBR)** is the equivalent of the MBR for the "text box" but it is not exactly the object as defined by MB (see "The MBR() of a text object", below). TB and TBR coincide in part only when the angle of the text is 0, but the width of the angle-0 TBR is not necessarily the width of the TB. If the angle is non-null, the TB is inscribed within the TBR, its nodes lying on the TBR sides.

TB is used explicitly for defining text objects with the "Create Text" command but is not "accessible" directly with a MB function even if MI remembers it when displaying a selected text. The only MB function giving some geographic information on a text object is ObjectGeography(obj, OBJ_GEO_MINX) [and _MINY, _MAXX, _MAXY] that returns the values used for building the TBR.

An overriding characteristic of text objects is the way MI handles the font point size of a text object. To be able to comprehend the import of this feature, let us look first how text objects can be created.

Text object creation.

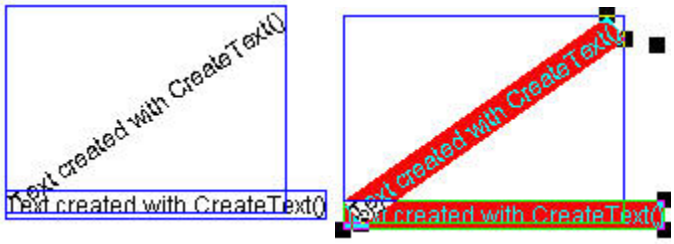
Such objects can be created directly in a map or in a layout with the text tool and its requesters or programmatically (in a MBX application or via the MapBasic window) with the function “CreateText()” or the command “Create Text”. There are some differences between the elements used for the creation and those stored in the object definition and some terminology used in the documentation may be misleading.

The **CreateText() function** requires the least information, essentially the text string, the angle with the horizontal, the reference point coordinates and the way the text is anchored to this reference point (in MB terms). The object is created directly into the ***editable layer*** of the specified window using the ***current font***.

The geometric definition of the object is completed by MB automatically: the size of the text box is calculated to fit the string with the current font characteristics (size, name and style in particular) and its position determined from the reference point and the relative anchoring of the text (“anchor” and “offset” like with the labels). Then the TBR for that text box set at the specified angle is calculated and stored in the object.

The two objects of the next figures were created with the following commands:

```
o=createtext(frontwindow(),2,55,"Text created with CreateText()",0,8,0)
o=createtext(frontwindow(),2,55,"Text created with CreateText()",35,8,0)
```

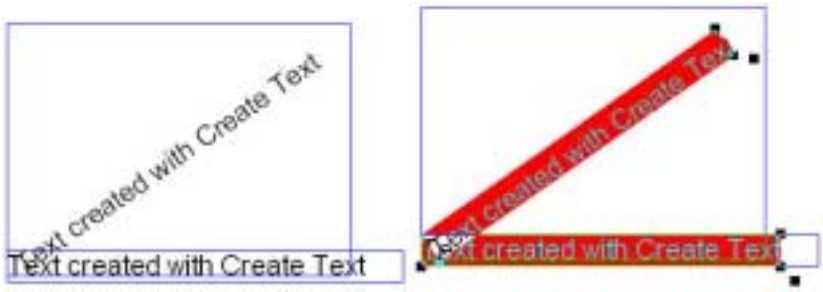


The TBR are in blue, the TB in red. The TB fit perfectly in the TBR
In the left diagram, the anchor point is invisible (two overlapping nodes cancel out in the display). To get that result, the parameter “anchor” is set to 8 (bottom right), other values displace the “rotation” center.

The **“Create Text” command** offers more possibilities (such as specifying a font different from the current one) but requires the size of the box into which the string will fit; the object can also be created in a window or in a variable. MI uses all the characteristics of the specified font or of the current font if none is specified, but for its size that is determined by the height of the specified box (the difference between the two Y coordinates).

The two objects of the next figures were created with the following commands:

```
create text into window frontwindow() "Text created with Create Text" (2,60) (8,59.5)
create text into window frontwindow() "Text created with Create Text" (2,60) (8,59.5) angle 35
```



The TBR are in blue, the TB in red. The TB is less wide than horizontal TBR because the horizontal TBR keeps the values specified in the command, here in excess of the required value.
In the left diagram, the anchor point is invisible (two overlapping nodes cancel out in the display)

> This example shows at least that the user does not have to worry about the actual size to give to the text box when dealing with strings on a slant. Another way to look at it is to imagine that MI creates all texts horizontally first, then modifies some of their geometry if the angle is non-null.

The **Text Tool** is a combination of the function and the command. It takes in only the anchor point coordinates (where the user clicked to position the text), uses by default the current font but offers also a requester for choosing a different one and takes in the point size of the font to determinate the size of the text box. Other parameters are line spacing, alignment (justify) and label line, all available only in the command and not in the function.

The creation of the object implies the translation of a font size given in points into geographic coordinates that will render the actual display size of a string dependent on the zoom (or scale) of the mapper. It also implies that the actual size of the text can be different for two objects created with the same font on the same mapper but at different zooms (or scales).

> I have not looked into the procedures/parameters used by MI to convert font size in geographic coordinates but it is not difficult to imagine a two-step operation; first a constant conversion from points to pixels, then a relative translation from pixels into height taking into account the value in “meters” of a pixel at the actual zoom of the mapper

Text object modification

Manual modifications using the Object Info requester are always possible and can be done on any part of the object definition.

Programmatically, modifications can be done using the “Alter Object” command and the following parameters. The general procedure would be to work on a selected object, alter it and update the table with the new version of the object

```
Dim o as object
o=selection.obj
Alter object o ...
Update selection set obj=o
```

Alter Object <o> INFO info_code, new_value

info_code	definition element
2	Font
3	Text string
4	Line spacing (1, 1.5 or 2)
5	Text justify (0 = left, 1 = center, 2 = right)
6	Label line (0 = no line, 1 = simple line, 2 = line with arrow)

Alter Object <o> GEOGRAPHY geo_code, new_value

geo_code	definition element
1	XMIN of TBR
2	YMIN ...
3	XMAX ...
4	YMAX ...
5	X coord of end of label line
6	Y coord of end of label line
7	Text angle

Note 1

The values defining the TBR can be changed only via the Alter Object command; there is no automatic readjustment of the values when other definition elements are changed such as line spacing; increasing the line spacing without changing the TBR height will result into a smaller font size but changing the font style to expanded will just extend the TB to the right of the TRB.

Note 2

To change the point size of an existing object one must act upon the height of the TBR. An approximate way to proceed is to think in terms of relative change to the height; if a change of x% is desired, then change the YMIN (or the YMAX) to change the difference (YMAX-YMIN) by x% (note that this technique will not work with text on a slant, alf<>0)

A more “precise” way to do it is to find the point size of the actual text object and to modify the “height” in order to obtain the desired point size. This procedure that hides many traps will be discussed in a special paragraph.

Note 2 a

Note 2 applies only to text objects in tables. When working in a layout, point size can be obtained from the objectinfo(o,2) function, specified in the Font argument of “Create Text” or modified with “Alter Object o INFO 2,makefont(...)”

Note 3

As some results of a modification using “Alter Object” can be uncertain, one must ask the question whether creating a new object to replace the old one would not be more appropriate. This issue is the object of a paragraph below.

The MBR() of a text object

If the MBR() of a text object is created with that function, it will not correspond to the rectangular box built with the MINX,MINY, MAXX and MAXY extracted from the object via ObjectGeography(). The following figure shows the MBR() outlined in RED and the rectangle from ObjectGeography(), the TBR, in blue.

One could also verify that the anchor point (upper left corner of highlighted TB) is the same as the upper left corner of the TBR and do not correspond to any position on the MBR object.

Conclusion:

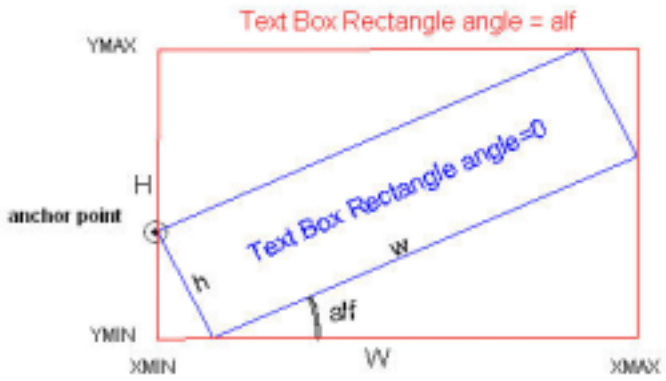
> ***when dealing with geometrics of text objects, never use values derived from the MBR(text_object) function.***

Practically, use a function such as TextBox() to build the Text Bounding Rectangle of a text object:

```
Declare Function TextBox(byval obj_txt as object, text_box as object)
    as logical

Function TextBox(byval obj_txt as object, text_box as object)as logical

if str$(objectinfo(obj_txt,1))<>"10" then
    textbox=0
    exit function
  else
    textbox=1
    create rect into variable text_box
        (objectgeography(obj_txt,1),objectgeography(obj_txt,2))
        (objectgeography(obj_txt,3),objectgeography(obj_txt,4))
end if

end function
```

The geometry of the TB/TBR pair

The TBR drawn on a text forming an angle with the horizontal is different from that of the same text perfectly horizontal, as can be seen in the next figure. The only constant is the XMIN side of the rectangle on which the anchor point is “grafted”, but several trigonometric equations can relate different elements of the geometric definition of the object.



Knowing the width W and height H of the TBR for a text with an “alf” angle, it is possible to deduce the width w and height h of the TBR for the same text in the horizontal position (alf=0). From the two simple expressions relating height and width of the two objects

$$H = w * \sin(alf) + h * \cos(alf)$$
$$W = w * \cos(alf) + h * \sin(alf)$$

we can find the values for the horizontal TBR

$$h = (H * \cos(alf) – W * \sin(alf)) / (\sin(alf)^2 – \cos(alf)^2)$$
$$w = (H * \sin(alf) – W * \cos(alf)) / (\sin(alf)^2 – \cos(alf)^2)$$

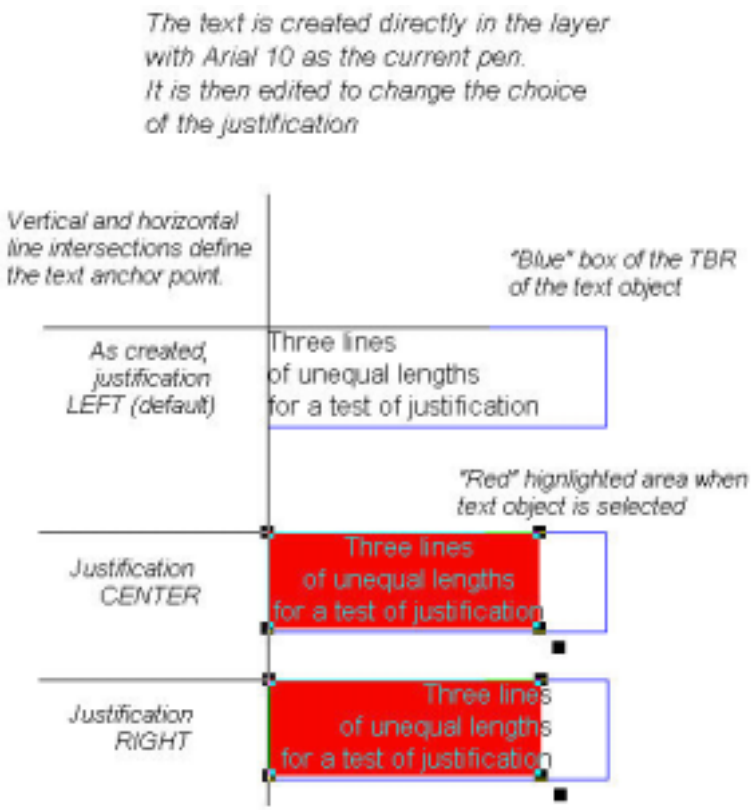
but these equations are valid only when the text angle alf is not a multiple of 45 degrees. In that case the denominator is null and the division forbidden. Geometrically, it means that the height (and the width) of the horizontal TBR is undefined, a wide range of values being acceptable.

The coordinates of the anchor point can be deduced from the 2 pairs of extreme coordinates. XANCH is simply XMIN. YANCH can be calculated from one of two formulas using the height and width of the horizontal TBR previously obtained.

$$XANCH = XMIN$$
$$YANCH = YMAX – w * \sin(alf) = YMIN + h * \cos(alf)$$

Justification in a multi-line text

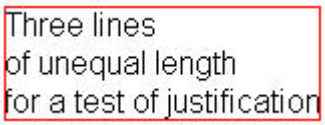
The justify property (left, center, right) comes into play only when the text spans more than one line. The following diagram shows that the alignment is made within the TB and not in the TBR



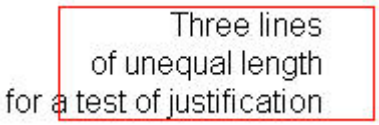
The text object in this diagram was generated by the “Create Text” command and that explains that the TBR is wider than the TB. If the CreateText() function is used, TBR and TB are identical.

```
o=createtext(frontwindow(),0,0,"Three lines"+chr$(10)+"of unequal length"+
chr$(10)+"for a test of justification",0,8,0)
```

yields that object:



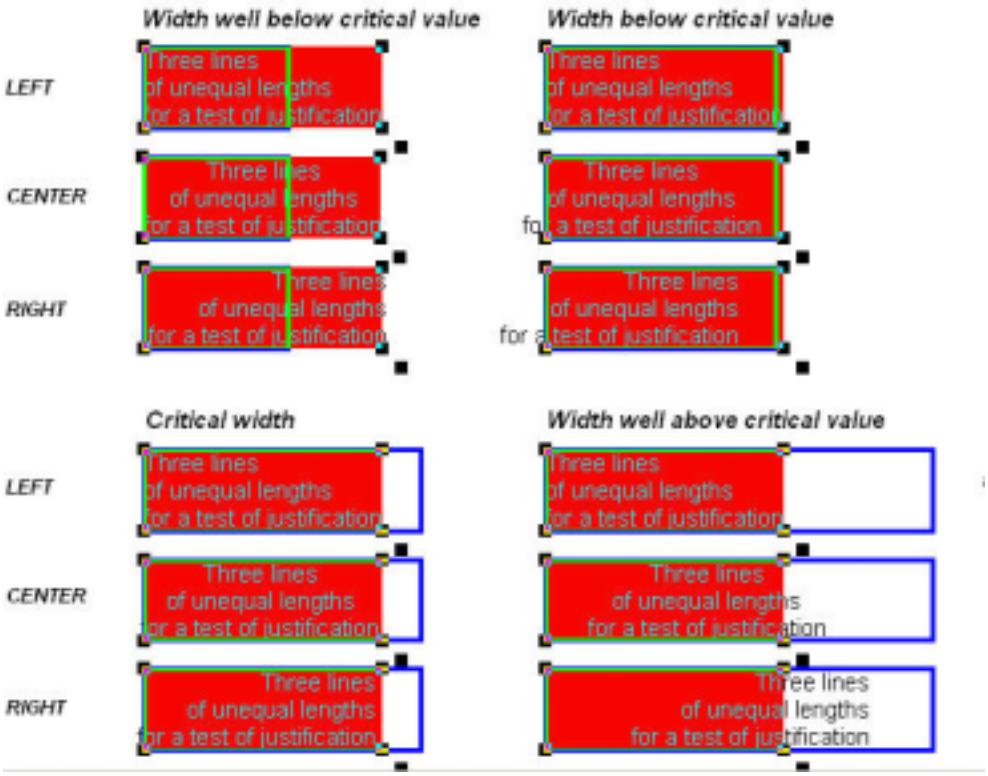
Setting justify to center or right can have unexpected results. If we change the last example to “justify left” (alter object o info 5,2), the text is not justified on the right side of the TMR but on some offset mark, as it can be seen on the next diagram



The following diagram shows the complexity of the results given different conditions

The blue boxes show the "Text Box Rectangle" of the text objects; their dimensions are exactly those given in the Create Text statement.

The highlighted select areas are the same whatever the size of the "Text Box". The "critical" width is that set by MI when creating the object with the Text Tool or the CreateText() function.

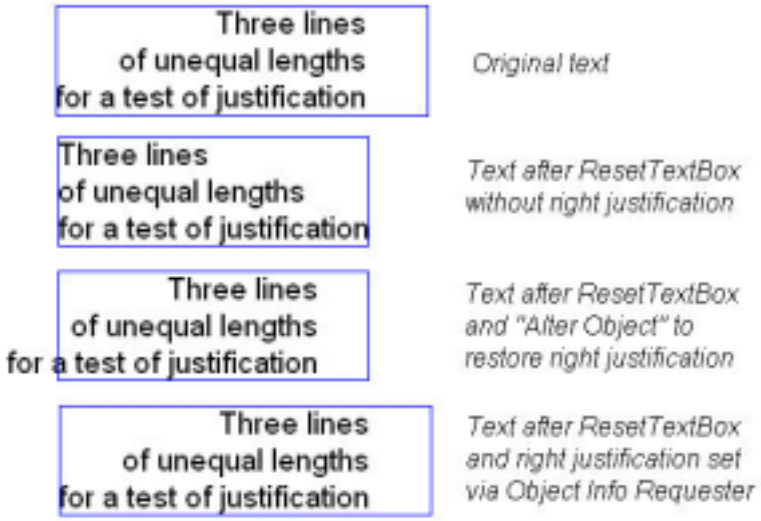


Experiences we carried out with the ResetTextBox() function (see appendix 2 “Resetting the TBR) under MI/MB 6.5 seem to indicate that the “Alter Object ... INFO 5, ...” command will destroy the fit of the TBR of text object obtained with the CreateText() function. (see next diagram)

Likewise, using the Object Info requester to edit its properties and changing the justification to right will create an offset from the right side of the box to the text. The TBR drawn in this circumstance gives the critical value of the diagram above, width necessary to enclose the entire text.

One must also notice that the anchor point of the text (XMIN, YMAX) is respected when the change is carried out with the requester but not with the Alter Object command (the left part of the text extends to the left of the anchor point).

The “right offset” that is created by the “justify right” is a nuisance for those who want to align the right end of several text objects on a vertical line (see the appropriate paragraph on the subject) if they count **more than one line and are right justified**.



"Blue" text boxes drawn with TestBox application

The value of that offset should be taken into consideration to adjust the anchor point of each text in order that the ends are aligned. We have some indications but no confirmation yet that this offset may be dependent of the font type, its size and its style. If all texts have the same characteristics, good alignment but on a approximate line position could be achieved; with mixed fonts, results are unpredictable.

Impact of a change in the TBR height

If we want to change the point size of a text object at a given zoom, we can modify the height of the TBR (e.g. by decreasing YMIN). If we do not change the width accordingly, we get some annoying results. Here is sequence of steps from the original text and its TBR (wider than required in this example).

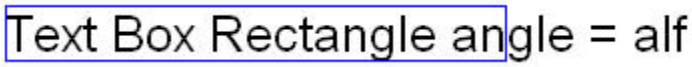
We alter the TBD height by giving a new value to YMIN by the traditional combination of commands

```
O=selection.obj
Alter object o geography 2, <new YMIN value>
Update selection set obj=O
```

That produces the new object looking like

If we deselect the object we get

We cannot see the object in full until we redraw (Ctrl-D) the window. The new TBR shown on the final image shows clearly that only its height has changed



There does seem to be an explanation to this strange behavior with texts wider than TBR that can be found at any time and not just after creation/modification. It is a situation that should be avoided as much as possible

Conclusion

> ***One should avoid using the "Create Text" command to create new text objects or the "Alter Object" to change the height of a text object if he does not control at the same time the width of the object***

Recommendation

> ***If one must use that technique, he should use a procedure to reset the TBR to its real value before inserting (or replacing) the new object in the table; see in appendix 2 "Resetting the TBR of a text object"***

Current and style requester fonts.

At opening of MI, the font style requester shows "Arial 9" and currentfont() returns the same value. If in the MBWindow, the "set style font makefont ("Arial",0,12,0,-1)" is run, the currentfont() is set to "Arial 12". However the font style requester remains at "Arial 9". The requester is updated to the currentfont if a text object is created using the Text Tool but remains unchanged if the object is created programmatically.

This relative independence of the requester and current styles has already been noticed; MI has provided an explanation that must be accepted. The users must be aware of the fact and make sure of the font that will be used in text creation if he does not want a bad surprise.

Modifying text size programmatically

There are two avenues open to modify a given text size programmatically, one is to modify the text object "height", the other is to recreate the object at the proper point size. Let us see the questions that they raise,

Modifying the text "height" by changing the bounds of the TBR with an "alter object" is the easiest method if the desired height of the text is known; the relative change between present and new can be applied to both height and width; even though the width change could yield results not exactly fitted to the text (there is no guarantee that for a specific string the ratio height/width is perfectly constant at every font size), it is

recommended to change it also in order to minimize the problems noted when the TBR is smaller than the TB.

> Multi line text with any kind of spacing can be treated that way as long as the angle is 0. We have seen that modifying the TBR bounds of a slanted text has no effect on the text size. ***This method will thus fail for slanted text***.

It is not possible to use the “alter object info 2,..” to **change the point size** because once the object has been created only its height in coordinate units is taken into consideration. To be able to act specifically on the point size, one must “recreate” the object using the CreateText() function and that implies playing with currentfont(), i.e. storing the present currentfont then setting it to the new values, creating the object with it, replacing it in the table and restoring the original currentfont.

But the CreateText() function does not deal with all the parameters of a text object. It will be necessary after its creation to modify the object with the remaining parameters (Line spacing, Label line and Justification)

If the new size is not provided, but instead a scaling factor is specified, the actual point size must be estimated before calculating the final one. (see special paragraph on “Estimation of a text object point size”)

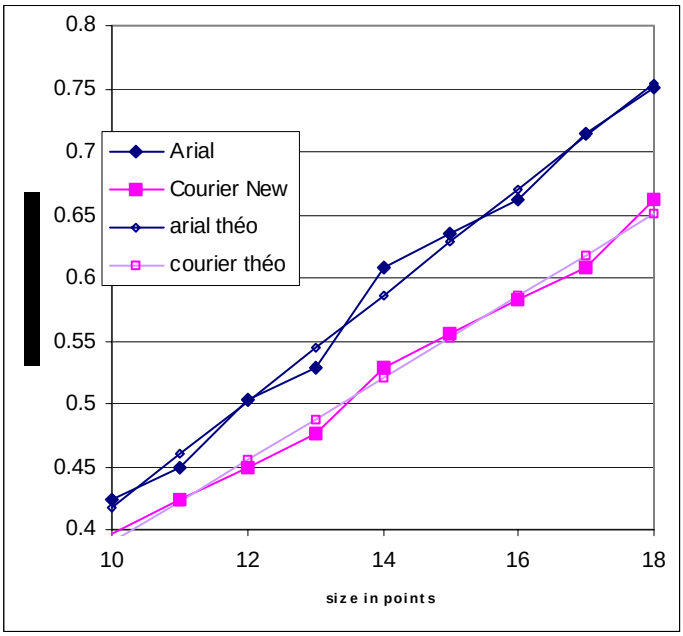
With all the methods, positioning of the “new” object relatively to the original one must be specified because it affects the relative position of the object in its surroundings. Even if all relative positions could be considered (i.e. the 9 “anchor” positions for labels) and that offset could also be added, most applications would be satisfied with two relative positions: should both object have the same “anchor point” (like in Upper Left), or the same centroid (like Centered)?

Estimation of a text object point size

There does not seem to universal equations that will transform a given point size of a font in a specified style into coordinate unit height and in the length of a string with the same font. MI converts points in units when the object is created and adapts the display of the string to the zoom of the mapper on the fly. MI translates also the height into points when the object is queried for that parameter. But in all cases MI treats point size as an integer and that is the source of some imprecision in the conversion of text height into font points.

As each font has its own spatial characteristics, it is not possible ti use a universal conversion table. The approach we have explored consist into finding a height/point conversion factor for the font in use with its actual style, then using it to find the actual point size of text and to calculate the “scaled” point size.

This technique is full of traps; it assumes first that there is a linear relationship between height and points, i.e. the conversion factor is constant. A short exploration will show that it is not so. Even if the range of point sizes is limited and only 2 fonts are tested, the differences to the straight regression line are most noticeable. Objects were created at the various point sizes with the CreateText() function, then their heights calculated with the YMAX and YMIN arguments.



If the principle of linearity is accepted, then one should estimate the parameters of the regression line over a large range of point sizes for the actual font with its style. An example of implementation is presented in the appendix 1 “From text height to point size” in the PointSizeEstimate() function.

Another source of imprecision exists for objects created with the Create Text command. The height specified by the opposite corners coordinates is converted into integer point values, hence the height might be larger than what could be required by the font of that point size.

All these comments justify the title of this topic: “Estimation of ...”. Short of more powerful MB functions, that aspect of dealing with text objects will remain an approximate job.

Alignment/spacing of text objects

This topic deals with the positioning of text objects one versus the others, and not about the justification of the lines within the same text object. Alignment requires the definition of a X coordinate that will be the reference on which text objects will be positioned; spacing requires a reference object and a vertical distance at which another object should be placed.

The procedure is simple: alter the object to be moved by modifying the “geography” arguments XMIN, YMIN, XMAX, YMAX with the appropriate values. For example, to get a left alignment on XLEFT: (the Y’s do not need to be altered)

```
XMINnu=XLEFT                    YMINnu=YMIN
XMAXnu=XLEFT+(XMAX-XMIN)        YMAXnu=YMAX
```

For right alignment on RIGHTX: (the Y’s do not need to be altered)

XMINnu=XRIGHT-(XMAX-XMIN) YMINnu=YMIN
XMAXnu=XRIGHT YMAXnu=YMAX

Given a reference object O_REF and a vertical distance DVERT (in the table coordinate units) the TBR of the next object to be set BELOW O_REF at the given spacing will be defined by: (the X's do not need to be altered)

XMINnu=XMIN YMINnu=ObjectInfo(O_REF,2)-DVERT
XMAXnu=XMAX YMAXnu= ObjectInfo(O_REF,4)-DVERT

Appendix 1 :

From text height to point size

This PointSizeEstimate() function generates 4 text objects using the same font and style as the one in the original object, with point sizes of 8, 16, 32 and 64. The coefficients of the regression line are used to estimate the point size of the original text from its height.

As the conversion applies to a single line of text, one must know the number of lines in the text. It is done with the CountTextLines() function that detects the number of Chr$(10) present in the text string.

If the object is slanted, the TBR dimensions of the “horizontal” text must me obtained because all the calculations must be done in that position. The function TextBoxSize() returns width and height of the object. There are several caveats associated with it: one is that the width returned is that of the TBR as defined at creation time, not necessarily, as we have seen, the actual width of the text. The other is the way I have solved the “forbidden” situation of 45 degrees slants by subtracting 0.00001 of a radian to the angle. For that particular situation, the result is more pure guessing than reality.

All these comments justify the title of this topic: “Estimation of ...”. Short of more powerful MB functions, that aspect of dealing with text objects will remain an approximate job.

```
'================================================================================
function PointSizeEstimate(byval obj_ori as object, pse as smallint) as logical
'================================================================================

dim o_tmp as object
dim f as font
dim h,verti,hori,s_p,s_p2,s_h,s_hp,a,b,nline as float
dim iret as logical
dim nlin as smallint

if str$(objectinfo(obj_ori,1))<>"10" then
    pointsizeestimate=0
    exit function
end if

f=objectinfo(obj_ori,2)

f=makefont(styleattr(f,1),styleattr(f,2),8,styleattr(f,4),styleattr(f,5))
set style font f
o_tmp = createtext(frontwindow(),objectgeography(obj_ori,1),
       objectgeography(obj_ori,2),objectinfo(obj_ori,3),0,3,0)
h=ObjectGeography(o_tmp,4)-ObjectGeography(o_tmp,2)
s_p=8
s_p2=8^2
s_h=h
s_hp=8*h

f=makefont(styleattr(f,1),styleattr(f,2),16,styleattr(f,4),styleattr(f,5))
set style font f
o_tmp = createtext(frontwindow(),objectgeography(obj_ori,1),
       objectgeography(obj_ori,2),objectinfo(obj_ori,3),0,3,0)
h=ObjectGeography(o_tmp,4)-ObjectGeography(o_tmp,2)
s_p=s_p+16
```

```
s_p2=s_p2+16^2
s_h=s_h+h
s_hp=s_hp+16*h

f=makefont(styleattr(f,1),styleattr(f,2),32,styleattr(f,4),styleattr(f,5))
set style font f
o_tmp = createtext(frontwindow(),objectgeography(obj_ori,1),
       objectgeography(obj_ori,2),objectinfo(obj_ori,3),0,3,0)
h=ObjectGeography(o_tmp,4)-ObjectGeography(o_tmp,2)
s_p=s_p+32
s_p2=s_p2+32^2
s_h=s_h+h
s_hp=s_hp+32*h

f=makefont(styleattr(f,1),styleattr(f,2),64,styleattr(f,4),styleattr(f,5))
set style font f
o_tmp = createtext(frontwindow(),objectgeography(obj_ori,1),
       objectgeography(obj_ori,2),objectinfo(obj_ori,3),0,3,0)
h=ObjectGeography(o_tmp,4)-ObjectGeography(o_tmp,2)
s_p=s_p+64
s_p2=s_p2+64^2
s_h=s_h+h
s_hp=s_hp+64*h

b=(s_h*s_p-4*S_hp)/(s_p^2-4*s_p2)
a=s_h/4 - b * s_p/4

iret=counttextlines(objectinfo(o_ori,3),nlin)
if objectgeography(o_ori,7)<>0 then
    iret=textboxsize(o_ori, hori, verti)
  else
    verti=ObjectGeography(o_ori,4)-ObjectGeography(o_ori,2)
end if
pse=int((verti-a)/b/nline)
pointsizeestimate=1

end function


'================================================================================
function CountTextLines(byval a_string as string,nlines as smallint) as logical
'================================================================================

dim ipos as smallint

counttextlines=0
ipos=0
nlines=1
boucle:
ipos=instr(ipos+1,a_string,chr$(10))
if ipos>0 then
nlines=nlines+1
goto boucle
end if
if nlines>0 then counttextlines=1 end if

end function
```

```
'================================================================================
function TextBoxSize(byval obj_txt as object, wid_txt as float,
       hei_txt as float) as logical
'================================================================================

dim wid_mbr,hei_mbr,ang_txt,wid_txt1,wid_txt2,hei_txt1,hei_txt2 as float

textboxsize=0
wid_mbr=abs(objectgeography(obj_txt,3)-objectgeography(obj_txt,1))
hei_mbr=abs(objectgeography(obj_txt,4)-objectgeography(obj_txt,2))
ang_txt=objectgeography(obj_txt,7)

if not ang_txt=any(45, 135, 225, 315) then
    ang_txt=ang_txt*0.01745329252
  else
    ang_txt=ang_txt*0.01745329252-0.00001
end if
wid_txt=(hei_mbr*sin(ang_txt) - wid_mbr*cos(ang_txt))/
       (sin(ang_txt)^2 - cos(ang_txt)^2)
hei_txt=(hei_mbr*cos(ang_txt) - wid_mbr*sin(ang_txt))/
       (cos(ang_txt)^2 - sin(ang_txt)^2)
textboxsize=1

end function

'================================================================================
```

Appendix 2 :

Resetting the TBR of a text object

The ResetTextBoxRect() function calls on first the PointSizeEstimate() function described in appendix 1 then the TextAnchorPoint() that returns the coordinates of the anchor point. With that information the object is recreated using the CreateText() function and alter to reintroduce some parameters not passed by that function (for the justification issue, see discussion in the main document)

```
'================================================================================

dim o, onu as object
dim iret as logical

o=selection.obj
set coordsys table selectioninfo(1)
iret=ResetTextBoxRect(o,onu)
if iret then
update selection set obj=onu

end if

'================================================================================
function ResetTextBoxRect(byval txt_ori as object,txt_rst as object) as logical
'================================================================================

dim pnts as smallint
dim f_cur, f_ori as font
dim xa,ya as float

ResetTextBoxRect=0
if str$(objectinfo(txt_ori,1))<>"10" then exit function end if
iret=PointSizeEstimate(txt_ori,pnts)
if not iret then exit function end if
f_cur=currentfont()
f_ori=objectinfo(txt_ori,2)
f_ori=makefont(styleattr(f_ori,1), styleattr(f_ori,2),
       pnts, styleattr(f_ori,4), styleattr(f_ori,5))
set style font f_ori
iret=TextAnchorPoint(txt_ori,xa,ya)
if not iret then exit function end if
txt_rst=createtext(frontwindow(), xa, ya, objectinfo(txt_ori,3),
objectgeography(txt_ori,7),8,0)
alter object txt_rst INFO 4,objectinfo(txt_ori,4)
alter object txt_rst INFO 5,objectinfo(txt_ori,5)
alter object txt_rst INFO 6,objectinfo(txt_ori,6)
ResetTextBoxRect=1

end function

'================================================================================
function TextAnchorPoint(byval obj_txt as object,
              Xanch as float, Yanch as float) as logical
'================================================================================

dim w,h,wid_mbr,hei_mbr,wid_txt,ang_txt as float

textanchorpoint=0
```

```
if str$(objectinfo(obj_txt,1))<>"10" then exit function end if
wid_mbr=abs(objectgeography(obj_txt,3)-objectgeography(obj_txt,1))
hei_mbr=abs(objectgeography(obj_txt,4)-objectgeography(obj_txt,2))
ang_txt=objectgeography(obj_txt,7)
if not ang_txt=any(45, 135, 225, 315) then
    ang_txt=ang_txt*0.01745329252
  else
    ang_txt=ang_txt*0.01745329252-0.00001
end if
wid_txt=(hei_mbr*sin(ang_txt) - wid_mbr*cos(ang_txt))/
           (sin(ang_txt)^2 - cos(ang_txt)^2)
Xanch=objectgeography(obj_txt,1)
Yanch=objectgeography(obj_txt,4) - wid_txt * sin(ang_txt)
textanchorpoint=1

end function

'=============================================================================
```