Global operations on objects within MapInfo.

Jacques Paris

jacques@paris-pc-gis.com

September 2001

Table of contents

The purpose of this document is to explore techniques to realize some global operations on groups of objects in ways useful to those who do not have access to MapBasic. That operation requires almost systematically the use of the MapBasic window to enter some expression based on the "Update …" command and to run it.[1] Many different results can be achieved quite immediately but some required data may have to be prepared first; most of the time the procedure will be automatic (transformation applied to all objects in a table) but processing can also be on a case-by-case basis

We will identify the functions that can be used in this context; they are essentially functions relating to characteristics of the object: shape, geography, contents, style. We will identify the nature of the operation, essentially the source of the data required and the scope of the transformation (all objects or a single object). We will end by exploring some applications of this systematized knowledge.

1 – GENERALITIES

1 – 1 Nature of an operation

Object manipulations cover the **creation** of objects from data contained in a table and **transformations** of existing objects on the basis of information contained in the original objects and sometimes from data contained in the table. These two aspects differ on several points; creating new objects calls on a limited set of resources, essentially the "create" functions, and draw the necessary data essentially from the table contents, while transformations include a much wider scope of resources; to the create "functions" are added other ones used to extract necessary information from already created objects. Creating new objects requires also adding specifications to the table definition, such as a coordinate system, that are already there for transforming objects.

1 – 2 Scope of an operation

The first characteristic of an operation is its scope. Let us consider first a **global** one: all the records in the specified table will be processed. Note that the table in question can be a entire base table or any selection of that table, the update being applied only to the objects of the table contained in that selection; we shall treat here a selection as a table.

As a typical example, let us consider the statement required to transform all the objects in a table to simple points representing their centroids.

```
Update my_table Set obj= createpoint(centroidx(obj), centroidy(obj))
```

One can see that the update will apply to all the objects that are in the table that has to be updated as it is the column "obj" that is updated and not the specific value of obj for one given object. That means that the transformation should draw some information from the object record i.e. from the columns values in the row of the object, including

[1] The <Table | Update Column…> menu command cannot not be used because its requester does not list "obj" as a possible column to be updated.

possibly the “original” value of obj; otherwise, the replacement object will be the same for all the cases in the “submitted” table.

The second observation we can draw from that example is the form of the expression “obj=…”; the = sign can be followed only by some function and precludes thus the use of any statement. Ex.: in MI/MB terminology, “CreatePoint()” is a function while “Create Point …” is a statement.

The scope can be case **specific**: the transformation will be applied to the case specified by its position in the table as recorded in the Rowid column. This should be considered different from the case of a selection containing only one object because with a selection, the expression does not point to a specific object but to the entire “selection” table. The same transformation as above applied to record 17 would be written:

```
Update my_table Set obj= createpoint(centroidx(obj), centroidy(obj)) where Rowid=17
```

1 – 3 Immediacy of an operation.

Some operation may be **immediate**, such as in the above example. It does not require any other data than that extracted from the original object (here the coordinates of its centroid).

In some cases, **intermediate** data must be inserted in the table to be able to carry out the transformation; that requires adding new columns, filling them up with data “extracted” from the objects and applying the transformation. This solution is sometimes chosen by those who wish to retain the “extracted” values for other uses. The transformation in the above example would require adding 2 columns (say Xcent and Ycent), updating them with the coordinates (one with centroidX(obj), the other with centroidY(obj)) and updating the obj column with

```
Update my_table Set obj= createpoint(Xcent, Ycent)
```

The addition of new data columns may not be necessary all the times, or even possible. As there is only one column holding objects in a table, one can consider creating **variables** in the MBW to hold temporary data for the transformation, but this is only applicable when dealing with one record at the time; the process cannot be “automated” for a whole table, repeating some chain of statements requiring always intervention by the user.

The “variable” solution is the only way to proceed when the objects must be modified using some statement like “Alter Object…” rather than radically transformed with some function like “createpoint()”. These remarks can be extended to the use of any pertinent statement that cannot be included in the “obj=…” expression. The commands would be something like:

```
Dim O as object
O=selection.obj
Alter object O ……
Update selection set obj=O
```

2 –BASIC PROCEDURES

2 – 1 Functions available for creating/transforming objects

There are four basic create functions: **CreateCircle()**, **CreateLine()**, **CreatePoint()** and **CreateText()**. These are the only ones available to create objects from scratch. All the other ones "transform" existing objects; I have identified the following ones in this category:

- two that switch back and forth the object type between pline and region: **ConvertToPline()**, **ConvertToRegion()**.

- three functions that return an object with a shape derived from the original one: **Buffer()**, **ConvexHull()** and **MBR()**,

- four other functions that require the presence of a second object to produce a transformed object: **Erase()** and its complementary **Overlap()**, **Combine()** and **OverlayNodes()** that simply adds nodes at intersection points of the two objects. In the context of this document, the second object will have to be under the form of an object variable because automatic processing can deal with one record, and thus one object, at the time; the object variable will of course be the same for all processed records.

2 – 2 Creating objects from tabular data

Only circle, point, line and text objects can be created directly from tabular data. The table is generally in a format other than MapInfo native format (delimited text, XLS, dbf…) and will contain columns with the necessary data:

Circle : (2 or 3 columns) x and y of center. The radius can be a constant entered by the user or be drawn from the third column.
Line : (4 columns) x and y for start and end points
Point : (2 columns) x and y
Text : (at least 3 columns, up to 6) x and y for anchor point, and text. Other parameters can also be read from the table or entered by the user: angle, position of text relative to anchor, offset.

Before creating the objects, some preparation is required.

1 – Open the file with [File | File Open] by choosing the proper **format** (delimited text, XLS …)

2 – Make the table **mappable** [Table | Maintenance | Table Structure (check "Table is mappable")]

3 – and in the same requester, choose the **projection** (it must be the same as that with which the data as been generated)

After creating, or modifying, objects do not forget to save the results [File | Save Table] or to abandon all the changes [File | Revert table].

If you want to view the results (before or after the save), [Window | New Map Window]

And to **create objects**, here are some examples:

Circle:

Values for radius are contained in column "col_for_radius"

```
Update my_table set obj=createcircle(xcent, ycent, col_for_radius)
```

or constant value of the radius (represented here by *999*)

```
Update my_table set obj=createcircle(xcent, ycent, 999)
```

Note 1: the value of radius is in CURRENT DISTANCE UNITS. If you are not too sure of what it is, run before the update (xxx is the abbreviated name of the unit)

```
Set Distance Units "xxx"
```

Note 2; the style of the created circles is the CURRENT BRUSH STYLE. Make sure that it is the correct one by checking with the right "button" for example.

Line:

```
Update my_table set obj=createline(xstart, ystart, xend, yend)
```

Note: the style of the created lines is the CURRENT PEN STYLE. Make sure that it is the correct one by checking with the right "button" for example.

Points:

```
Update my_table set obj=createpoint(xcent, ycent)
```

Note: the style of the created points is the CURRENT SYMBOL STYLE. Make sure that it is the correct one by checking with the right "button" for example.

Texts:

Update my_table set obj=createtext(*window_id*, xanchor, yanchor,col_for_text,0,0,0)

Note 1; the texts are created in a specific map window; *window_id* can be replaced by frontwindow() if the texts are for the active window.

Note 2: the last three zeroes represent in sequence
- the angle (horizontal 0, to 360, counter clockwise)
- the position relative to the anchor (0 center, 1 top left, 2 top center, 3 top right, 4 center left, 5 center right, 6 bottom left, 7 bottom center, 8 bottom right)
- the offset (from 0 to 50 points) from relative position 1 to 8

Values can be assigned for any or all or read from appropriate columns.

Note 3: the style of the created text is the CURRENT FONT STYLE. Make sure that it is the correct one by checking with the right “button” for example.

2 – 3 Transforming objects

When transforming objects, the safest way is to work with a copy of the original table; very often indeed the results of the transformation will be used jointly with the original map; very often also the transformation will be applied to a part of the objects in the original table that can selected by their column contents or their types[2]. It is very appropriate then to make a selection of the objects and save the selection and not the original table: one can expect an appreciable gain of file space and possibly of displaying speed.

There is an alternative to making a copy before starting that I consider less safe: when work is completed, use [File | Save Copy As] with a new table name followed by a [File | Revert Table] to eliminate all the changes from the original. The lack of safety resides in making a [File | Save Table] instead of a [File | Save Copy As] and in the risk of interruption or of system failure when working on the transformations.

With these considerations taken into account, we will concentrate on the operations themselves. The basic principle is to replace an original object by another one with some or all of its characteristics taken from the original and possibly others entered by the user (an identical value applicable to all the objects) or extracted from a column, as in the two cases for the radius in the CreateCircle() above.

To be able to extract characteristics from an object, we must call on various functions. I have organized them in different functional families:

Note that the same argument value can have different interpretations according to the type of the queried object particularly with ObjectGeography()

Structure of the objects: (functions often called for by others)

Objectinfo(obj,20)	*total number of nodes in a polyline or a region*
Objectinfo(obj,21)	*number of sections in a polyline or of polygons in a region*
Objectinfo(obj,21+N)	*number of nodes in the Nth section or polygon*

Coordinates - nodes:

CentroidX	(obj)	*x of the centroid of an object*
CentroidY	(obj)	*y of the centroid of an object*
ObjectGeography(obj,1)		*x of a point object* *x of start node of line* *xmin of MBR of any other object type*
	(obj,2)	*y of a point object*

[2] If the selection is made on object type, see the appendix on that subject

		y of start node of line
		ymin of MBR of any other object type
	(obj,3)	*x of end node of line*
		xmax of MBR of any other object type
	(obj,4)	*y of start node of line*
		ymax of MBR of any other object type

ObjectNodeX	(obj,1,1)	*x of starting node of a polyline or region*
ObjectNodeY	(obj,1,1)	*y of starting node of a polyline or region*
ObjectNodeX	(obj,objectinfo(obj,21),objectinfo(obj,21+objectinfo(obj,21)))	
		x of ending node of a polyline or region
ObjectNodeY	(obj,objectinfo(obj,21),objectinfo(obj,21+objectinfo(obj,21)))	
		y of ending node of a polyline or region

note: the use of objectinfo() is required to take into account the possibility that a polyline may be composed of several sections or a region of several polygons. If one is sure that there are no "multiple" objects, the expressions for ending node can be simplified to

(obj, 1, objectinfo(obj,20))

special geometric information:

ObjectGeography(obj, 5)	*beginning angle of Arc object*
ObjectGeography(obj, 6)	*ending angle of Arc object*

ObjectGeography(obj, 5)	*diameter of circle used for rounding corners of a rounded rectangle object. In coordinate units.*

text information:

ObjectGeography(obj, 5)		*x of end of label line (text object)*
	(obj, 6)	*y of end of label line (text object)*
	(obj, 7)	*angle of the text with horizontal*
ObjectInfo	(obj, 3)	*string representing the body of a text object*
	(obj, 4)	*line spacing*
	(obj, 5)	*centering of text*
	(obj. 6)	*label line style*

object dimensions:

Area	(obj, "xxx")	*area of a closed object, xxx is one of MI area units*
ObjectLen	(obj, "yyy")	*length of a linear object, yyy is one of MI distance units*
Perimeter	(obj, "yyy")	*perimeter of a closed object, yyy …id …*

Specifying the projection

If the transformation is immediate, the projection does not have to be specified; that would be the case when no coordinates must be stored in a tabular form. But it the transformation requires first to extract some coordinates from the objects and place them

in some columns of the table, or to read coordinates values already in the table, the projection must be given. To do so, write and run fom the MapBasic window

```
Set Coordsys table my_table
```

2 – 4 Direct replacement of objects

Objects are replaced by objects of a different type. New objects can be:

points (symbols), applicable to all types

```
Update my_table Set obj=CreatePoint(CentroidX(obj), CentroidY(obj))
```

lines (2 nodes), applicable to one-section polylines

```
Update my_table Set obj=CreateLine(ExtractNodeX(obj,1,1), ExtractNodeY(obj,1,1),
  ExtractNodeX(obj,1,objectinfo(obj,20)), ExtractNodeY(obj,1,objectinfo(obj,20))
```

circles, applicable to all types

```
Update my_table set obj=createcircle(centroidX(obj), centroidY(obj), radius)
```

Radius a constant, a column name or an expression.

buffers, applicable to all types excluding texts[3]

```
Update my_table Set obj=Buffer(obj, res, width, d.u.)
```

res number of nodes in a circle (used for rounded ends and elbows)
width buffer width in distance unit
If width is <0 and the object is a closed object (region…,), the newly created object will be smaller than the original. If the negative value is so large that no buffer can be created (no area), the operation will yield an unspecified error
d.u. one of MI distance unit (e.g. “m”)

MBRs, minimum bounding rectangle, applicable to all
MBR width/height of point object = 0

```
Update my_table Set obj=MBR(obj)
```

[3] The direct update using the buffer() function does not accept text while the Create Object As Buffer statement does!!!

2 – 5 Extracting data from objects into tabular form

Data that is contained in the object definition must be sometimes transformed in tabular data. The procedure is straightforward:

1 – create the columns required to hold the data in the table. Make sure that they are of the right type and the right size.

2 – make sure that the coordinate system in use is the right one. Run from the MapBasic window

set coordsys table *your_table*

3 – update each column with the appropriate “function”

4 – save the table when finished.

The following table contains all the pertinent information.

Nature of the information	**Column definition**	**Function**	**Applicable to**
Object Type	SmallInt	ObjectInfo(obj,1)	All types
Centroid X	Float	CentroidX(obj)	All types
Centroid Y	Float	CentroidY(obj)	All types
Point X		ObjectGeography(obj,1)	Points(=CentroidX)
Point Y		ObjectGeography(obj,2)	Points(=CentroidY)
MBR X mini	Float	ObjectGeography(obj,1)	All types
MBR Y mini	Float	ObjectGeography(obj,2)	All types
MBR X maxi	Float	ObjectGeography(obj,3)	All types
MBR Y maxi	Float	ObjectGeography(obj,4)	All types
Line beginning X	Float	ObjectGeography(obj,1)	Lines
Line beginning Y	Float	ObjectGeography(obj,2)	Lines
Line ending X	Float	ObjectGeography(obj,3)	Lines
Line ending Y	Float	ObjectGeography(obj,4)	Lines
Number of nodes	Integer	ObjectInfo(obj,20)	Plines, Regions
Number of sections/polygons	Integer	ObjectInfo(obj,21)	Plines, Regions
Number of nodes in Nth section/polygon	Integer	ObjectInfo(obj,21+N)	Plines, Regions
First node X of Nth section/polygon	Float	ObjectNodeX(obj,N,1)	Plines, Regions
First node Y of Nth section/polygon	Float	ObjectNodeY(obj,N,1)	Plines, Regions
Last node X of Nth section/polygon	Float	ObjectNodeX(obj,N, ObjectInfo(21+N))	Plines, Regions
Last node Y of Nth section/polygon	Float	ObjectNodeY(obj,N, ObjectInfo(21+N))	Plines, Regions
Area	Float	Area(obj,”*area unit*”)	Closed objects
Object Length	Float	ObjectLen(obj,”*distance_unit*”)	Linear objects

Perimeter	Float	Perimeter(obj,"*distance_unit*")	Closed objects
Beginning Angle	Float	ObjectGeography(obj,5)	Arc objects
Ending Angle	Float	ObjectGeography(obj,6)	Arc objects
Rounding diameter	Float	ObjectGeography(obj,5)	Rounded rectangle
Text string	String	ObjectInfo(obj,3)	Text objects
Line spacing	Float	ObjectInfo(obj,4)	Text objects
Justification of text	SmallInt	ObjectInfo(obj,5)	Text objects
Label line style	SmallInt	ObjectInfo(obj,6)	Text objects
Label line end X	Float	ObjectGeography(obj,5)	Text objects
Label line end Y	Float	ObjectGeography(obj,6)	Text objects
Angle of text	Float	ObjectGeography(obj,7)	Text objects

(*) maxi 254; when specifying the width, make sure the longest text will not be truncated

One could also derive some information for an ellipse object. Theoretically its main axes should be equal to the sides of the MBR rectangle. There are however some discrepancies between MBR derived values and those obtained by double clicking on the object. There seems to be major differences particularly with non-projected maps. One should proceed with caution unless more definitive information is obtained on that subject.

3 – EXAMPLES OF IMPLEMENTATION

3 – 1 Two complex transformations

Task 1

All objects must be transformed in rectangles of given W and H centered of the objects centroids.

Solution: build the MBR of a line representing the rectangle diagonal

```
Update my_table Set obj=MBR(CreateLine(CentroidX(obj)-W/2, CentroidY(obj)-H/2,
                    CentroidX(obj)+W/2, CentroidY(obj)-H/2))
```

Task 2

Transform region objects in circles with areas equal to region areas

Solution: obtain the region area with the area() function and convert it in the radius of the circle.

```
update my_table set obj=createcircle(centroidx(obj),centroidy(obj),sqr(area(obj,"sq mi")/3.14159))
```

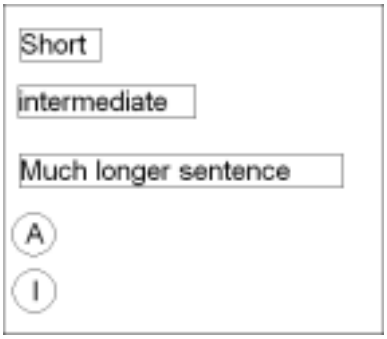
note: the area unit used in the area() function must be in accordance with the distance unit presently in use. One may have to "force" the right unit to be used with a

Set Distance Unit "*unit abbrev.*"

3 – 2 Framing text objects

It is easy to frame text objects; the texts are in one layer and the frames in another, or all are in the same layer but the frame brush must be transparent for the text to appear in all circumstances. The frames can be built directly by a MBR transformation (sharp corners), the buffer transformation being unavailable for text objects from the update requester. One could also include circles among the possible frames with the difference that circles are used usually with fixed radius while rectangular frames thus created are automatically adjusted to word length.

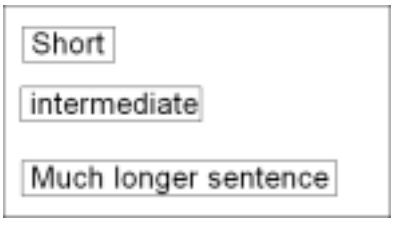
The direct results are however not satisfactory because the MBR is usually oversized (the longer the text, the wider and the higher the rectangle) and aligned with the left of the text. This lack of symmetry is less obvious with circles that can be used for single letters or digits quite effectively.



The longer-than-the-text box and the off-centering can be corrected empirically by using a longer update formula with correction factors. If we use:

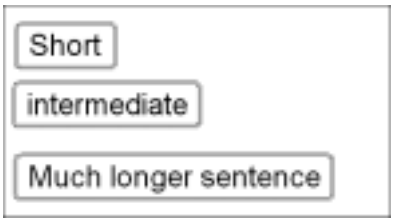
```
update selection set obj=mbr(createline(
objectgeography(obj,1)-(objectgeography(obj,4)-(objectgeography(obj,2))/4,
objectgeography(obj,2),
objectgeography(obj,1)+(objectgeography(obj,3)-objectgeography(obj,1))*0.86+
(objectgeography(obj,4)-objectgeography(obj,2))/4,
objectgeography(obj,4)))
```

The xmini is shifted to the left by an adjustable constant equal to one quarter of the height of the box. The xmax is first shifted to the left by an adjustable percentage of the box length (here 14%=1.0-0.86) then to the right by the same quarter of the box height. Results without being perfect are certainly more appealing than the first ones.



More manipulations are now possible, including making rounded corners frames with the buffer() function from the previous rectangles

update selection set obj=buffer(obj,20,2,"m")



There is a major limitation to framing texts that way: it is not possible to deal directly with texts on an angle. MBR do not represent the rectangle in which the text itself would be contained as if the angle was 0, but that in which the rotated text is inscribed; it is therefore not a good indicator of the size of the text. Heavy calculations are required to obtained from that MBR and the rotation angle [objectgeography(obj,7)] the size of the actual text rectangle that could be used to build a first rectangle, converted into region and rotated. It is too much to expect from the update command.

3 – 3 Moving simple objects around

This technique does not work will polylines and regions because these objects are defined by their nodes rather than their MBR. It shows the use of an intermediate variable and how to set up a repetitive operation in the MapBasic window. The first 3 lines are run once:

```
dim o as object
dim irow as integer
fetch first from my_table
```

The next lines are written in, highlighted and run as a block by a succession of <enter>. When no object moves anymore, the end of the table is reached. The 1000 value is the lateral displacement in meters in the MTM projection map I used; it could be negative. If a vertical displacement was required, two more alter object statement will be needed with indices 2 and 4 (instead of 1 and 3) and with the value of the Y displacement instead of 1000.

```
irow= my_table.rowid
o= my_table.obj
alter object o geography 1 , objectgeography(o,1)+1000
alter object o geography 3 , objectgeography(o,3)+1000
update my_table set obj=o where rowid=irow
fetch next from my_table
```

3 – 4 Using a closed object to split regions

With that example, we will be emulating complex operations. The purpose of the exercise is to replace a group of objects with their parts split with a unique closed object.

The splitting object must first be defined then selected and is stored in an object variable called O for simplicity. In the MapBasic window:

```
Dim O as object
O=selection.obj
```

The table containing the objects to be processed must be editable. Select the objects to split, copy and paste them, then

```
Update selection set obj=erase(obj,O)
```

Select the objects to split again, copy and paste them, then

```
Update selection set obj=overlap(obj,O)
```

Select the objects that have been split and delete them not before making sure that the tabular data of the new objects is correct. Pack the table.

3 – 5 Resetting centroids

Centroids of region may sometimes be located in what appears to be a bad position, generally along the border. For example, if a region is created by starting with a “null” region (no nodes) then adding one node at the time with the “Alter Object Node Add …” statement, the centroid remains stuck with the last added node. An automatic reset of the centroids in the positions MI calculates can be done by running these two lines one after the other:

```
Update my_table set obj=ConvertToPline(obj)
Update my_table set obj=ConvertToRegion(obj)
```