The Alter MapInfoDialog Statement

Documentation assembled by

Jacques Paris

with the contribution of

Lars I. Nielsen

May 2006

The “Alter MapInfoDialog” statement

The “Alter MapInfoDialog” statement that exists in MapBasic since 3.0, the first MI version for Windows “disables, hides or assign new values to controls in MapInfo’s standard dialog boxes” (MapBasic Reference Guide, v6.5 p 37).

Follows a double warning that changes in MI versions may imply changes in those standard dialogs (dialog ID, contents and controls ID) and that different Windows versions may have an impact on “Common Dialogs” based Windows resources.

The syntax of this statement is double, to set new settings:

```
Alter MapInfoDialog  dialog_ID
          Control  control_ID
               { Disable | Hide  | Value  new_value  } [ ,  { Disable...  } ]
          [  Control... ]
```

and to restore default settings:

```
Alter MapInfoDialog  dialog_ID  Default
```

This little talked about MB statement hides many little treasures but requires some investment in the identification of the vital *dialog_ID* and *control_ID* data, some knowledge about rules and constraints governing its use and some ideas of potential application areas.

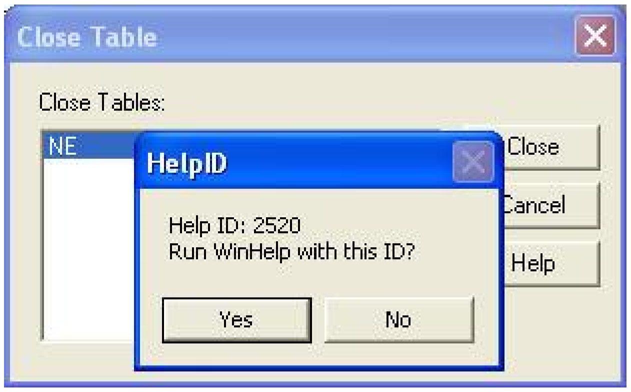
The purpose of this document is three-pronged; it provides means to identify the specific IDs for a given MI version and even for any of its regionalized sister-versions; it reviews the rules and shows what can be done with this statement; it identifies general areas of potential application and outlines solutions to some real life situations.

> NOTE: some dialogs that are used in normal MI operations do not seem to be “standard” dialogs; they do no exist in MIRES... and thus cannot be accessed with the Alter statement. They are open by direct menu actions but their item ID do not work in a Run Menu Command from the MBW. This is the case, for example, of all the Graph Formatting menu commands (MI 7.0).

What is available and how to fetch it.

The alter MapInfoDialog statements requires knowing the ID of a dialog and those of its controls. As you have MapInfo, you can easily obtain the ID of an open dialog;

you just have to start MapInfo with “-helpdiag” in the command string[1]. When a dialog is open, clicking on the dialog “Help” button opens a small requester such as the shown in the following image.

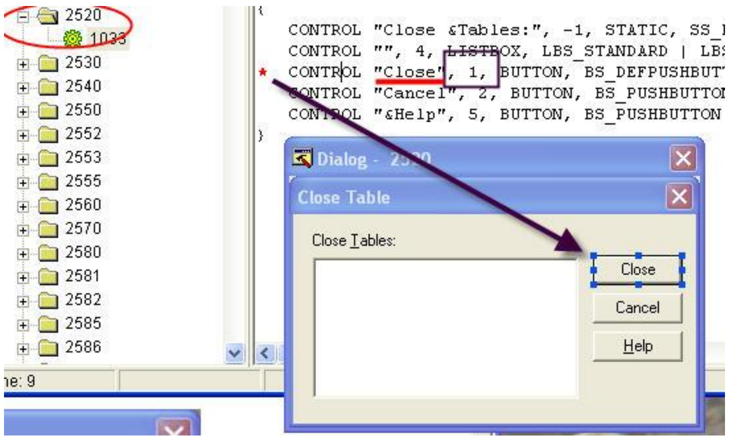


Help ID: 2520
Run WinHelp with this ID?

The ID for the “Close Table” dialog is 2520

To identify the controls ID, one must then use some generally free software such as WinId (disadvantage of displaying all values in hexa) or Windose. But both these programs cannot provide the MI dialog IDs (hence the use of MI as a first step) and they, as well as MapInfo with “-helpdiag”, will work only on OPEN dialogs.

I recommend rather using a little marvel: Resource Hacker[2] that reads all the resources in a dll and reveal ID for dialogs and controls. It can also produce RC files (code for generating resources) that Lars I. Nielsen uses for rebuilding dialogs with all the necessary IDs (see Lars’ output in separate MIproXXX_en_dialogs.zip files).



```
{
    CONTROL "Close &Tables:", -1, STATIC, SS_
    CONTROL "", 4, LISTBOX, LBS_STANDARD | LBS
    CONTROL "Close", 1, BUTTON, BS_DEFPUSHBUT
    CONTROL "Cancel", 2, BUTTON, BS_PUSHBUTTO
    CONTROL "&Help", 5, BUTTON, BS_PUSHBUTTON
}
```

RH displays the code for building the dialog and the dialog itself. We can see here the “Close Table” dialog (ID 2520, in the list of resources to the left and in the title bar under the black arrow). The correspondence between code and graphic is very

1 A word of warning: with the specific setup MI7.0 + Windows XP (my desktop), MI hangs at closing time (program not responding, requiring using task manager) but with MI6.5 and Windows 2000 (my laptop) there is no problem. Do not be surprised if you get the same problem.
[2] Resource Hacher is a free (for non-commercial use) software that can be downloaded from http://rpi.net.au/~ajohnson/resourcehacker

clear: the line with a red star corresponds to the control outlined in blue. In that line after the title ("Close") can be found the control ID (1, framed). A value of '-1' (as in the first line) simply means that no ID was assigned to that control.

There is a limit to what Resource Hacker and Lars's rebuilt dialogs can do; they work only for the MI standard dialogs listed in MIRESxxx.dll[3] and cannot find Common windows-based dialogs. These are not stored as real dialogs; they are created within MI from common resources (such as ComDlg32.dll) and modified by the equivalent of the "Alter MapInfoDialog" statement and/or using some partial overlays. I have identified some of these Common Dialogs that are presented in a separate document.

Re-settable settings

All accessible controls (= controls with an ID) can be reset by the keywords Hide and Disable; thus, controls that are registered in an MI dialog as hidden or disabled cannot be revealed or activated as the keywords Show and Enable are not recognized by the statement.

All controls can also be reset with a *new_value* but the nature of that parameter is different for the various types of control.

Control type	Setting	Variable	Details
Button OKButton CancelButton	title	string	If the string is longer than the button width, it is chopped at both ends as it is centre aligned
Statictext	title	string	If the string is longer than the button width, it is chopped at the right end as it is left aligned
Editbox	value	string	
Listbox	value	integer	Selects by its position the item of the list that should be displayed (selected) The list of items cannot be changed The items of some listboxes may include checkboxes (like in the toolpad options dlg). These checkboxes cannot be modified.

3 Resource Hacker and Lars' tools can certainly work on different dlls, such the ComDlg32, but their dialogs are not used as such by MI, and the information thus gained would essentially documentary and not much operational.

Multilistbox	value	integer	Selects by its position the first item of the list that should be displayed (selected) The list of items cannot be changed and only one Item of the list can be selected
PopupMenu	value	integer	Selects by its position the item of the list that should be displayed The list of items cannot be changed
Checkbox	value	0 or 1	0 clears the box, 1 checks it
Radiogroup			A radiogroup is coded by as many separate controls as there are options in the radiogroup. 1 as *new_value* in a control selects it. Only one control can be thus selected; more specifications will not be considered and may even prevent any change.
"Style" Picker	value	Return value of a Make… function	Symbol, Pen, Brush, Font or Pen&Brush (region style) can be given new values such as those produced by the appropriate Make…() function.
Groupbox			Purely cosmetic; no interest for this paper

General areas of potential use

It could be useful to recap here how the statement works. When it is run with its new values, the dialog is altered but remains hidden. To reveal it, one must reach the point where a menu item (or possibly a sequence of dialogs) will display it, or one can use the "run menu command" with the appropriate code. If the dialog contains only pre-fixed parameters (options), it does not have to be displayed to be "effective". It is only when new data is entered to be processed that the dialog must be displayed because processing will take place only when the OK button is activated. Because of that constraint, the "alter Midlg" statement cannot be used in an automated loop; it will always require user's intervention.

Three general headings cover the range of potential uses:

- setting general parameters i.e. controlling the global MI environment,
    - An example are the preferences that can be set by the sub-items of that menu (printer, startup,…) and cannot be accessed with the MapBasic language

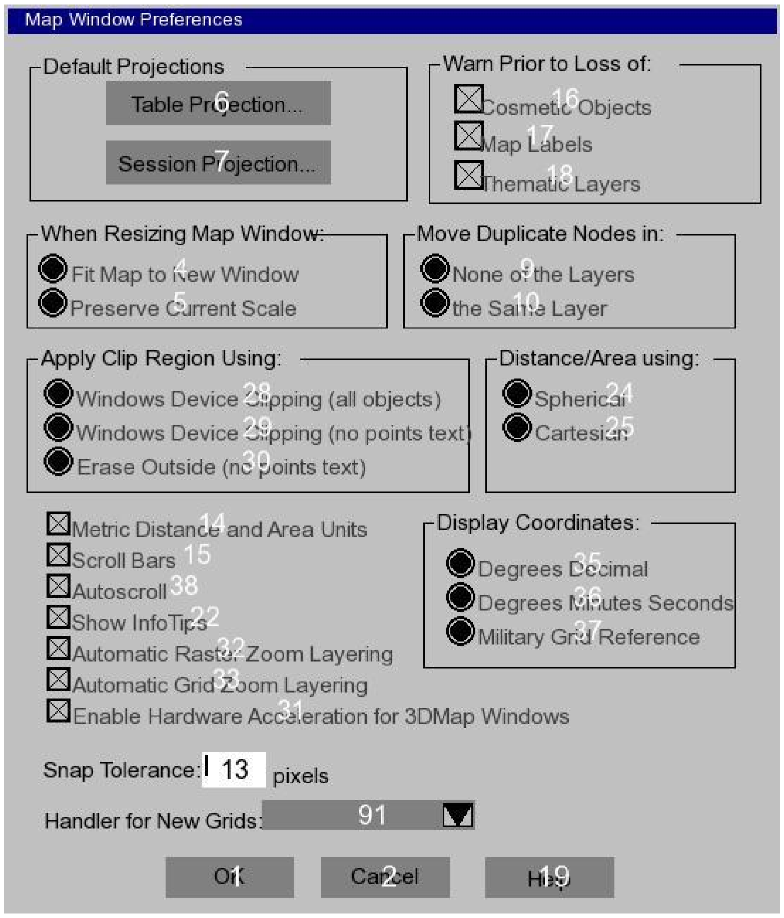
- setting local parameters i.e. modifying the specific work environment,
    - options for specific operating environment, such as the various "new" windows (Map, Browser, Layout,…) when created. MapBasic statements

of the “set …” family can modify those windows only after they are created.

- data entry for processing i.e. introducing some “variable” data
  - some dialogs include data entry controls (edittext) that accept “variable” data that will be used to perform the “operation”, as with the “find” dialog where an address can be input to be “found” on the map or any object info from an editable layer (coordinates, style). There are generally MapBasic statements to take care of these operations.

Examples of implementation

1 - Setting Map Window Preferences (general parameters)



These parameters preside at the creation of each new mapper window. They cannot be modified “globally” by MB statements even if some may be adjusted for a specific (open) mapper window as part of the map Options (see the “Set map” statement)..

We find in this dialog (apart from the classic Ok, Cancel and Help buttons, 1, 2 &19)

- two buttons (6, 7) calling the same dialog (1870, Choose Projection)
- 2 blocks of 3 (16, 17, 18) and 7 (14, 15, …) checkboxes
- 3 radiogroups with 2 choices and 2 with 3
- an editbox (13) to enter the number of pixels
- one popup menu (91)

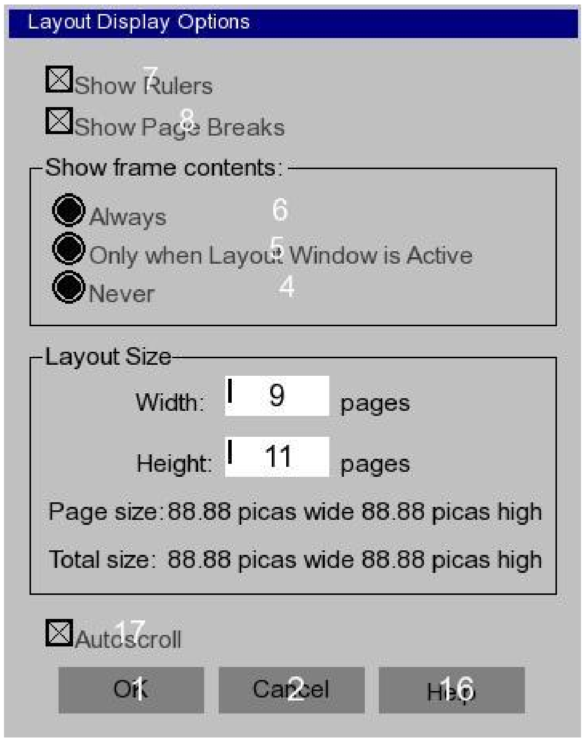
Our choices are to preserve current scale (5), to use Cartesian calculations (25), not to autoscroll (38) and to have a snap tolerance of 5 pixels. The alter statement will look like

```
Alter MapInfoDialog 4130    control 5 value 1    control 25 value 1
                            control 38 value 0   control 13 value 5
```

2 - Setting the layout window (local parameters)

We have first to find the nature of the existing controls.

- 7, 8 and 17 are checkboxes
- 6, 5, and 4 form one radiogroup (as there are as many controls ID as ‘buttons’, these will require special handling)
- 9 and 11 are edittext boxes
- 10, 12, 13, 14 and 1553 are statictext and of no interest for our purpose
- 1, 2 and 16 buttons



Layout Display Options

☒ Show Rulers
☒ Show Page Breaks

Show frame contents:
● Always
● Only when Layout Window is Active
● Never

Layout Size
Width: 9 pages
Height: 11 pages
Page size: 88.88 picas wide 88.88 picas high
Total size: 88.88 picas wide 88.88 picas high

☒ Autoscroll

OK Cancel Help

All these types of control can be hidden (keyword Hide) or disabled (keyword Disable) but once these keywords have been used for a control, the only way to return it to its original state is to reset the entire dialog to “default”; the Alter statement does not recognize the Show or Enable keywords.

My personal options:

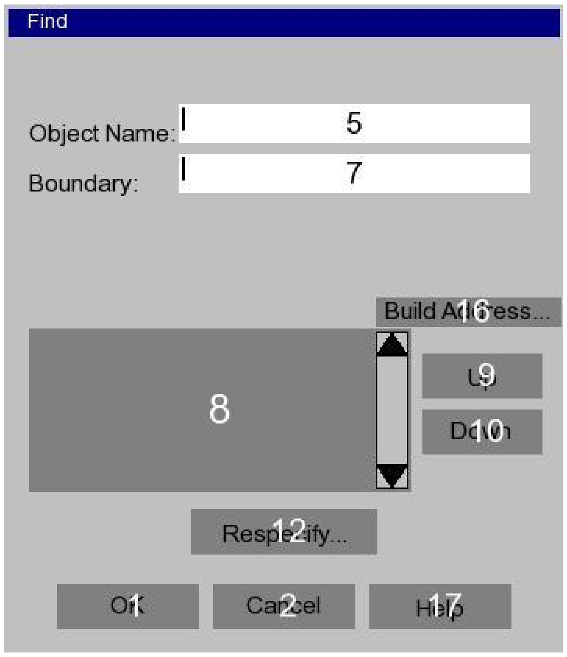
I want to see the rulers (7) but not the page breaks (8); I want the contents to always show (6) and a layout size de 2*2 pages (9, 11). I want to make sure there is no autoscroll (17) and I do not want to see Help (16)

```
Alter MapInfoDialog 1550      control 7 value 1        control 8 value 0
                              control 6 value 1        control 9 value “2”
                              control 11 value “2”     control 17 value 0
                              control 16 Hide
```

3 - Processing data

MI gives a very simple example in the MB User’s Guide (v6.5), finding an address. The dialog ID 2202 is an example of several

situations one finds in a dialog that does not look always as the stored (and displayed here) version: the “boundary” + editbox ID 7 appear only if a boundary table has been selected for refining the search, and the control ID 16 is never to be seen



What we are interested in is the editbox to enter the “address” ID 5. To enter a new one, then we execute

Alter MapInfoDialog 2202 control 5 value “new_addr”

That in itself in not enough; the dialog has been modified but for the search to take place, it must be “run”, hence a new step

Run menu command M_ANALYZE_FIND
Or
Run menu command 305

That command will open the dialog but to run it, one must activate the OK button.

What could be the advantage of such a technique over entering data directly into the editbox? It has to do with the automation of a find process. We can easily imagine that we have a list of addresses in some file or table. We can easily write a small MB code to read one address at the time into a string variable and use that variable as the new “value”. Then in the same loop, alter the dialog and run it.

For each new address that solution requires the intervention of the user and that total supervision of the coding is the strong point and only justification of that technique. As we are writing code, we could use instead the Find statement with its Interactive keyword. However with this statement, the dialog would appear only if user’s intervention is required to clear a doubt, i.e. when a choice must be made between possible solutions. That small difference may be judged important enough to promote the “alter MIdlg“ solution.

As a perceived safeguard, MI proposes also to hide the Respecify button (ID 12) to limit the risks that the user would change the search definition (table, column) and modify thus the chances to find the addresses.

Alter MapInfoDialog 2202 control 12 Hide

If we consider working in loop, that statement must be run once before the loop beginning, the button reappearing only after an “Alter MapInfoDialog 2202 default” is executed (e.g. after loop has ended).