REGIONMANAGER

By Ron Kaplan

This document created in December 2021, last edited September 2023.

Medley comes equipped with a core set of functions for specifying regions and creating the windows that occupy those regions on the screen. But it can be disruptive if not irritating to have to draw out a new ghost region for every invocation of a particular application. Thus the common applications (e.g. TEDIT, SEDIT, DINFO...) implement particular strategies to reduce the number of times that a user has to sweep out a new region. They instead default to regions that were allocated for earlier invocations that are no longer active. TEDIT for example recycles the region of a session that was recently shut down, SEDIT allocates from a list of previous regions, DINFO always uses the same region, but FILEBROWSER always prompts for a new one. Applications that do recycle their regions tend to do so indiscrimately, without regard to the current arrangement of other windows on the screen or the role that those windows may play in higher-level applications.

The REGIONMANAGER package provides simple extensions to the core region and window functions. These are aimed at giving users and application implementors more flexible and systematic control over the specification and reuse of screen regions. It introduces three new notions:

A "typed region" allows the regions of particular applications to be specified, classified, and recycled according to their types.

The size, location, and orientation of a "relative region" is specified with respect to particular screen points and the location of other windows.

A "constellation region" encloses the collection of satellite windows (prompts, menus, etc) that surround the central window of an application.

REGIONMANAGER is innocuous in that explicit user action is required to change the default behavior of any system components.

Typed regions

REGIONMANAGER adds overlay veneers to the core `CREATEW`, `CLOSEW`, and `GETREGION` functions to make it easier to predict and control how different applications arrange their windows on the screen without always needing to respond to a ghost-region prompt.

The `REGION/INITREGION` arguments may now be region-type atoms in addition to either NIL or particular regions as `CREATEW` and `GETREGION` otherwise allow. The type-atom will resolve to a region drawn from a predefined pool of regions associated with that type, if the pool has at least one that is not currently allocated to another window. If the pool has no available regions, then the pool will be enlarged with a region that the user produces from a normal ghost-region prompt, and the type-atom will then resolve to the newly installed region.

A typed-region is marked as "inuse" and therefore unavailable when `CREATEW` assigns it to a window, and the extended CLOSEW marks it as again available when the window is closed. The region of the most recently closed window will be offered the next time a region of its type is requested.

An example of how an application can take advantage of this facility is the TEDIT-PF-SEE package. This provides lightweight alternatives to the PF and SEE commands that print their output to scrollable read-only Tedit windows, specifying PF-TEDIT and SEE-TEDIT as their region types. The user can predefine a preference-ordered sequence of recyclable regions that bring up multiple output windows in a predictable tiled arrangement, without region-prompting for each invocation.

The global variable TYPED-REGIONS is an alist that maintains the relationship between atomic type-names and the list of regions that belong to each type. The list is ordered according to preferences set by the user, and a type-atom is always resolved to the first unused region in its list. If the user is asked to sweep out a new region, that region is added at the end, as the least preferable. The function SET-TYPED-REGIONS is provided to add or replace TYPED-REGION entries.

(SET-TYPED-REGIONS TYPELISTS REPLACE) [Function]

TYPELISTS is an alist of the form

((type$_1$. regions$_1$)(type$_2$. regions$_2$)...)

where each regions$_i$ is a possibly empty list of regions. For convenience, if TYPELISTS is just a literal type-atom, it is interpreted as ((type)), and if it is a list (type . regions) begining with an atom, it is interpreted as ((type . regions). The new regions replace preexisting regions if REPLACE, otherwise they are added at the front.

Typically, a call to SET-TYPED-REGIONS would be placed in a user's INIT file to set up the preference order for the regions that the user wants to participate in this reallocation scheme. If an application uses a type that is not on TYPED-REGIONS, then that type-atom is treated as NIL and always gives rise to the normal ghost-region prompting. Thus a user will observe no change in system behavior if TYPED-REGIONS is left with its initial value NIL. A type that is added with an empty region list (as opposed to not being on the list at all) will allow new regions to accumulate for recycling.

The function REGION-TYPE returns NIL if X is not a typed-region or not a region of type TYPE.

(REGION-TYPE X TYPE) [Function]

In most scenarios the interpretation of a typed region specification is handled automatically by the extended CREATEW and GETREGION functions. Sometimes it may be useful to perform to for the regions dimensions to be entered into other calculations before it is installed in a window. The function GRAB-TYPED-REGION recycles an existing REGION-TYPE window if one meets the optional minimum width and height requirements, otherwise a new region is returned.

(GRAB-TYPED-REGION REGION-TYPE MINWIDTH MINHEIGHT) [Function]

A type can be assigned to an untyped region and installed in a window by the function REGISTER-TYPED-REGION. That region will then be recycled when the window is closed.

(REGISTER-TYPED-REGION REGION REGION-TYPE WINDOW) [Function]

If REGION is NIL, the (presumably) untyped region of WINDOW will be registered. An entry in TYPED-REGIONS will be created for REGION-TYPE if it is not already present.

Relative regions

Two functions are provided to make it easy to create regions relative and oriented with respect to a specified reference point. These may be useful for constructing an application that includes a constellation of windows arranged in a particular relative way.

(RELCREATEREGION WIDTH HEIGHT CORNERX CORNERY REFX REFY ONSCREEN) [Function]

RELCREATEREGION creates a region of dimensions WIDTH and HEIGHT. One of its corners is identified by CORNERX and CORNERY and that corner will be aligned with a reference screen-point determined by REFX and REFY. If ONSCREEN, the WIDTH or HEIGHT will be adjusted with respect to that alignment so that the resulting region is entirely within the screen.

WIDTH and HEIGHT can be given as absolute (natural) numbers or specified relative to the WIDTH and HEIGHT of another region or of the screen. The possibilities are interpreted as follows:

natural number: the number of screen points

list of the form (anchor fraction adjustment), where anchor is a region, window, or an atom SCREEN or TTY. The corresponding dimension of the anchor is mutiplied by `fraction` and `adjustment` is added to the result. For example, specifying `(<window> .5 -1)` results in a WIDTH that is one point smaller than half the width of window's region. `Fraction` and `adjustment` default to 1 and 0 respectively.

region/window/SCREEN/TTY: equivalent to (region/window/SCREEN/TTY 1 0).

CORNERX can be LEFT, RIGHT, or NIL=LEFT, CORNERY can be BOTTOM, TOP, or NIL=BOTTOM. If LEFT/TOP are specified, for example, the region will be displayed down and to the right of the reference point. If RIGHT/BOTTOM, then up and to the left.

The reference-point arguments REFX and REFY are interpreted as follows:

NIL: LASTMOUSEX/LASTMOUSEY

natural number: an absolute screen coordinate

(anchor fraction adjustment) or just region/window/SCREEN/TTY: the quantity determined relative to the size of anchor (as above) is added to the anchors left/bottom produce the REFX/REFY coordinate. In this case, fractions specified as LEFT/BOTTOM/NIL are interpreted as 0 and RIGHT/TOP are interpreted as 1. For example, a specification (<window> `.4` -2) for REFY will produce a coordinate 2 points below the level that is 40% of the distance between the bottom and top of the window's region.

For convenience, if REFX is a position and REFY is NIL, then the XCOORD and YCOORD of REFX are taken as absolute values for REFX and REFY.

Also for convenience, if WIDTH is a potentially a list of RELCREATEREGION arguments, then the elements of that list are spread out in a recursive call.

`(RELGETREGION WIDTH HEIGHT CORNERX CORNERY REFX REFY MINSIZE)` [Function]

Calls GETREGION with an initial ghost region as created by RELCREATEREGION. CORNERX and CORNERY determine the ghost region's fixed corner, and the cursor starts at the region's diagonally opposite corner. If MINSIZE is true, then WIDTH and HEIGHT are taken as the minimum sizes of the region, except for adjustments that may be needed to ensure that all corners of the ghost region are initially visible on the screen.

`(RELCREATEPOSITION REFX REFY)` [Function]

Creates a position with X and Y coordinates specified by REFX and REFY references as above.

Constellation regions

Applications are often set up as a constellation of windows, a central or primary window surrounded by some number of satellites for menus, headers, prompts, and secondary outputs. The main panel of a file browser, for example, displays the list of files, but above it are carefully arranged windows for the column headers, summary information, and prompts, and off to the side is the menu of file browser commands. FILEBROWSER interprets the screen region that the user sweeps out for a new browser as the region for the whole constellation,the smallest region that will enclose the central window and all of its satellites. Similarly, the screen region given to TEDIT and SEDIT is divided between the prompt window and the central editing window, again so that the whole constellation (a pair in these cases) fit within the provided region.

Each of these applications is constructed by anticipating the subregions that the satellite windows will occupy after they are attached, decreasing the constellation region by their estimated (using WIDTHIFWINDOW HEIGHTIFWINDOW) or actual sizes, and then using remainder as the region for the central window.

An alternative approach is to construct the central window first, giving it the entire constellation region, and then to have ATTACHWINDOW reshape that window to accomodate the satellite windows as they are attached in sequence. This leads to the same final configuration, but there is no need for separate calculations to pre-adjust the region of the central window.

REGIONMANAGER provides an overlay veneer for ATTACHWINDOW that implements this strategy. If the new argument TAKEFROMCENTRAL is true, then the region of the WINDOWTOATTACH will be substracted from the region of the existing central window according to the EDGE parameter of the attachment.

(ATTACHWINDOW WINDOWTOATTACH MAINWINDOW EDGE POSITIONONEDGE WINDOWCOMACTION TAKEFROMCENTRAL)[Function]

This behavior is also triggered if the UNDERCONSTRUCTION property of the central window is true. Thus, a constellation can be set up by creating all of the satellites and the central window, marking the central window as under construction, and then doing the sequence of attachments. The property can be reset to NIL when the construction is complete, so the central window does not shrink if other windows are attached (e.g. expanded menus) by later user actions.

A somewhat weaker form of a constellation is a collection of windows that are not attached around a central window but stand in a parent-child relationship at least with respect to closing and moving. A parent windows spawns children that respond independently to ordinary window commands (move, shape, close). But the children close when the parent closes, and the children move when the parent moves so that they continue to appear in the same relative positions. These primitives allow the construction of a tree of windows that are dependent in this way.

(CLOSEWITH CHILDREN PARENT [Function]

Establishes a link between the PARENT window and any number of CHILDREN windows such that all CHILDREN will close when PARENT closes. The closing is accomplished by CLOSEWITH.DOIT:

(CLOSEWITH.DOIT PARENT) [Function]

Closes the close-with children of PARENT.

(MOVEWITH CHILDREN PARENT) [Function]

Establishes a link between the PARENT window and any number of CHILDREN windows such that all CHILDREN will move when PARENT closes. The closing is accomplished by MOVEWITH.DOIT:

(MOVEWITH.DOIT PARENT NEWPOS) [Function]

If NEWPOS is the new position of PARENT, moves each of the move-children so that they stand in the same relation to PARENT after it moves as before.