## Freemenu Internal Documentation

All names are in the Freemenu package (except old names, which begin with \\fm.)

### The Freemenu Description Language:

A freemenu description is defined by:

| | |
|---|---|
| [] | optional |
| {} | group of things for some other operator |
| * | 0 or more |
| + | 1 or more |
| \| | or |
| | **literals in bold** |

menu-desc --> menu-element ; can be passed to **freemenu**

menu-element --> item-desc | group-desc

item-desc --> **(**{item-prop value}* **:label** item-label {item-prop value}***)**

item-prop --> one of the prop keywords described in the user doc

item-label --> string | bitmap | imageobj

group-desc --> **(**[**:group**] [prop-spec] {menu-element}***)**

prop-spec --> **(:prop** {group-prop value}***)**

group-prop --> one of the group property keywords described in the user doc

### Group Formats:

The possible formats are:

:column - the elements in this group are layed out vertically, top to bottom, flush left

:row - the elements in this group are layed out horizontally, left to right, along the same baseline

:table - the elements in this group are layed out vertically, top to bottom, flush left. additionally, the second element of each direct subgroup is positioned at the same horizontal location. the third element of each direct subgroup is positioned at the same horizontal location. and so on, such that each element of each subgroup is both in a row and a column.

:explicit - each element of this group has a :left and :bottom property specifying its position. if the group property :coordinates is :group, then the values of the :left and :bottom property are relative to the lower left corner of the group, and if :menu (the default) they are relative to the lower left corner of the topmost group.

### Group Format Defaults:

The format of a group of menu-elements can be specified in the prop-spec for that group. If it is not, the following rules apply:

- The default format for a group-desc passed directly to **freemenu** is :column.

- When formatting a group in :column format, the default format of each sub group is :row.

- When formatting a group in :row format, the default format of each sub group is :column.

- When formatting a group in :explicit format, the default format of each sub group is :explicit

- When formatting a group in :table format, the default format of each sub group is :table-element, which signals the formatter to horizontally align each element in the sub group with the other elements in the table. If the format is specified for a sub group, the elements of that sub group will not be aligned with the other elements in the table.

## The Formatter:

### Entry point:

The entry point to the freemenu formatter is \\fm.format (fm::format). It takes a description of the menu to be formatted and returns a group hierarchy structure. In the current version of freemenu, the group structure is just an alist of group id's and properties, with the topmost group first. The formatter takes the following arguments:

description : a menu-element as defined above

format : the format to be used to lay out this group

font : the default font for each item in this group, must be a fontdescriptor

left, bottom : the lower left corner of the group. format everything relative to this position.

rowspace : the number of pixels to leave between rows, that is, the space to leave between elements in a column

columnspace : the number of pixels to leave between columns, that is, the space to leave between elements in a row

mother : the mother group of the one being passed for formatting. in the current version, this is the ID of the mother group, not the group itself.

The remaining arguments are optional. In the current version they are not specified, but they are SET by the guy who processes the group prop-spec. They might want to be specified in later versions:

id - the ID to use for this group

props - the prop-spec to use for this group

**Group prop-spec:**

The macro \\fm.setupprops processes the group prop-spec in the description and fills in the slots in group accordingly (currently plist format). At the same time, it sets the arguments above from the values specified in the prop-spec, thus overriding the passed-in default. When the formatter is done with the current group, the function return, and thus all of the arguments are popped off the stack, and the previous state is now dynamically visible. This is how the formatter keeps track of format prop state, and pops back to the previous state when done formatting a group.

\\fm.setupprops takes a group-spec and a list of group props to set. It generates code that will fill in the group props information, and then sets the format state arguments for the props that are in the list of group props to set.

The :left and :bottom group props in the prop-spec specify offsets for the entire group from where the formatter would otherwise position the group. This is similar to the way :left and :bottom item props specify offsets for an item that is automatically formatted (as opposed to explicitely positioned in the menu description).

**Group boxing:**

The macro \\fm.checkforbox looks in the props to see if the group is boxed, and if it is, it adjusts the left,bottom position of the group to allow for the width of the box and the boxspace (boxoffset = box-width + box-space).

Then the elements in the group are layed out normally.

Finally, the macro \\fm.updateforbox, if the group is boxed, does the following: save the calculated extent of the group as the interior region of the group, and then adjusts the region to include the box, and saves this region as the region property of the group.

**Layout routines:**

The formatter calls one of the layout routines (\\fm.layout-column \\fm.layout-row \\fm.layout-table \\fm.layout-explicit) to lay out the elements in the group. The layout routines provide the real guts of the formatting. These functions return multiple values:
1. list of items in all the elements layed out
2. list of groups in all the elements layed out
3. extent region of all the elements layed out
4. list of id's of all the subgroups layed out (this one would go away if had real group structure, instead of flat alist)

**Layout algorithm:**

Here is a description of the algorithm used in laying out a column of elements. The other routines follow the same procedures, but operate in a different dimension or have other overhead (like layout-table).

```
bind extent ; the region the elements in this group occupy
        itemlist ; a flattened list of items in this group
        grouplist ; a flattened list of groups in this group
        subgroupids ; a list of id's of groups in this group
        element-position ; the position of the next element formatted
        element ; temp for hanging onto the element newly formatted

iterate through each element in the group description
  if the element is a group description:
        set element to call format on the group description
        extend: extent, itemlist, grouplist, subgroupids with the results
  otherwise:
        set element to create an item from the item description
        extend extent, itemlist with the item

  increment element-position by the size of element and extra space

return extent, itemlist, grouplist, subgroupids
```

**Putting it all together:**

With the information returned by the layout routine in the hands of the formatter, create a new group structure from the group prop-spec and extent, itemlist, and subgroupids. Add this group to the front of the list of groups layed out, and you get a list of groups for the description just formatted.

**Freemenu Data Structures:**

A Freemenu is currently a window, with all of the necessary properties set to make it behave as a menu when it is open. Eventually a Freemenu probably wants to be an independent structure, which can be enclosed in different display mechanisms, like windows, image-objects, pop up managers, etc.

There are three main data structures composing a Freemenu:

ITEM - Instance of the datatype freemenuitem, one for each item in the menu. The macro itemprop provides access to fetching and replacing fields in the datatype. The macro %itemprop is an internal version of the same macro which doesn't type check the item and requires the field (property) name to be provided explicitely (not bound to some variable) in the call.

GROUP - A list structure describing a group of items in the menu. List fomat is (<group-id> <group-type-identifier> {<prop> <value>}*). The <group-id> is used for ASSOC purposes on the list of all groups. The <group-type-identifier> is checked by the macro group-p to ensure that this list is a valid freemenu group. The macro groupprop provides access to getting and setting the props in the cddr of the list.

NWAY - A list structure describing an nway collection in the menu. The list format is the same as that for groups. nway-p and nwayprop are analogous to group-p and

groupprop.

A list of all the items in the menu is stored on the ITEMS window property.

A list of all the groups in the menu is stored on the GROUPS window property.

A list of all the nway collections in the menu is stored on the NWAYS window property.

The functions get-item, get-group, and get-nway take an id and search the appropriate list for a matching freemenu item, group, or nway, respectively.

Additionally, many of the freemenu functions, like redisplay-menu, depend on being able to use a flat list of objects in the menu, either items, groups, or nway collections.