The Customization Library

This manual describes how to declare customization groups, variables, and faces. It doesn't contain any examples, but please look at the file '`cus-edit.el`' which contains many declarations you can learn from.

All the customization declarations can be changes by keyword arguments. Groups, variables, and faces all share these common keywords:

`:group` value should be a customization group. Add symbol to that group.

`:link` value should be a widget type. Add value to the external links for this customization option. Useful widget types include `custom-manual`, `info-link`, and `url-link`.

`:load` Add value to the files that should be loaded before displaying this customization option. The value should be either a string, which should be a string which will be loaded with `load-library` unless present in `load-history`, or a symbol which will be loaded with `require`.

`:tag` Value should be a short string used for identifying the option in customization menus and buffers. By default the tag will be automatically created from the options name.

Declaring Groups

Use `defgroup` to declare new customization groups.

defgroup symbol members doc[keyword value]... Function

Declare symbol as a customization group containing members. symbol does not need to be quoted.

doc is the group documentation.

members should be an alist of the form ((name widget)...) where name is a symbol and widget is a widget for editing that symbol. Useful widgets are `custom-variable` for editing variables, `custom-face` for editing faces, and `custom-group` for editing groups.

Internally, custom uses the symbol property `custom-group` to keep track of the group members, and `group-documentation` for the documentation string.

The following additional keyword's are defined:

`:prefix` value should be a string. If the string is a prefix for the name of a member of the group, that prefix will be ignored when creating a tag for that member.

Declaring Variables

Use `defcustom` to declare user editable variables.

defcustom symbol value doc[keyword value]... Function

Declare symbol as a customizable variable that defaults to value. Neither symbol nor value needs to be quoted. If symbol is not already bound, initialize it to value.

doc is the variable documentation.

The following additional keyword's are defined:

`:type` value should be a widget type.

`:options` value should be a list of possible members of the specified type. For hooks, this is a list of function names.

`:initialize`
: value should be a function used to initialize the variable. It takes two arguments, the symbol and value given in the `defcustom` call. Some predefined functions are:

  `custom-initialize-set`
  : Use the `:set` method to initialize the variable. Do not initialize it if already bound. This is the default `:initialize` method.

  `custom-initialize-default`
  : Always use `set-default` to initialize the variable, even if a `:set` method has been specified.

  `custom-initialize-reset`
  : If the variable is already bound, reset it by calling the `:set` method with the value returned by the `:get` method.

  `custom-initialize-changed`
  : Like `custom-initialize-reset`, but use `set-default` to initialize the variable if it is not bound and has not been set already.

`:set`
: value should be a function to set the value of the symbol. It takes two arguments, the symbol to set and the value to give it. The default is `set-default`.

`:get`
: value should be a function to extract the value of symbol. The function takes one argument, a symbol, and should return the current value for that symbol. The default is `default-value`.

`:require`
: value should be a feature symbol. Each feature will be required when the 'defcustom' is evaluated, or when Emacs is started if the user has saved this option.

See section "Sexp Types" in The Widget Library , for information about widgets to use together with the `:type` keyword.

Internally, custom uses the symbol property `custom-type` to keep track of the variables type, `standard-value` for the program specified default value, `saved-value` for a value saved by the user, and `variable-documentation` for the documentation string.

Use `custom-add-option` to specify that a specific function is useful as an member of a hook.

custom-add-option symbol option Function
: To the variable symbol add option.

  If symbol is a hook variable, option should be a hook member. For other types variables, the effect is undefined."

Declaring Faces

Faces are declared with `defface`.

de ace face spec do[keyword value]... Function
: Declare face as a customizable face that defaults to spec face does not need to be quoted.

  If face has been set with 'custom-set-face', set the face attributes as specified by that function, otherwise set the face attributes according to spec

  doc is the face documentation.

  spec should be an alist of the form '`((display atts)...)`'.

atts is a list of face attributes and their values. The possible attributes are defined in the variable 'custom-face-attributes'.

The atts of the first entry in spec where the display matches the frame should take effect in that frame. display can either be the symbol 't', which will match all frames, or an alist of the form '((req item...)...)'

For the display to match a FRAME, the req property of the frame must match one of the item. The following req are defined:

`type` (the value of (window-system))
Should be one of `x` or `tty`.

`class` (the frame's color support)
Should be one of `color`, `grayscale`, or `mono`.

`background`
(what color is used for the background text)
Should be one of `light` or `dark`.

Internally, custom uses the symbol property `face-defface-spec` for the program specified default face properties, `saved-face` for properties saved by the user, and `face-documentation` for the documentation string.

Usage for Package Authors

The recommended usage for the author of a typical emacs lisp package is to create one group identifying the package, and make all user options and faces members of that group. If the package has more than around 20 such options, they should be divided into a number of subgroups, with each subgroup being member of the top level group.

The top level group for the package should itself be member of one or more of the standard customization groups. There exists a group for each *finder* keyword. Press C-h p to see a list of finder keywords, and add you group to each of them, using the `:group` keyword.

Utilities

These utilities can come in handy when adding customization support.

custom-manual Widget
Widget type for specifying the info manual entry for a customization option. It takes one argument, an info address.

custom-add-to-group group member widget Function
To existing group add a new member of type widget, If there already is an entry for that member, overwrite it.

custom-add-link symbol widget Function
To the custom option symbol add the link widget.

custom-add-load symbol load Function
To the custom option symbol add the dependency load. load should be either a library file name, or a feature name.

customize-menu-create symbol &optional name Function
Create menu for customization group symbol. If optional name is given, use that as the name of the menu. Otherwise the menu will be named 'Customize'. The menu is in a format applicable to `easy-menu-define`.

The Init File

Customizations are saved to the file specified by `custom-file`, as calls to `custom-set-variables` and `custom-set-faces`.

When you save customizations, the current implementation removes the calls to `custom-set-variables` and `custom-set-faces`, and replaces them with code generated on the basis of the current customization state in Emacs.

By default `custom-file` is your '`.emacs`' file (for GNU Emacs and older XEmacs) and is '`custom.el`' in the same directory as '`init.el`' (in XEmacs 21.4 and later). If you use another file, you must explicitly load it yourself.

As of XEmacs 21.4.7, when `custom-file` is present, it is loaded *after* '`init.el`'. This is likely to change in the future, because (1) actions in '`init.el`' often would like to depend on customizations for consistent appearance and (2) Custom is quite brutal about enforcing its idea of the correct values at initialization.

Wishlist

- Better support for keyboard operations in the customize buffer.
- Integrate with '`w3`' so you can get customization buffers with much better formatting. I'm thinking about adding a <custom>name</custom> tag. The latest w3 have some support for this, so come up with a convincing example.
- Add an 'examples' section, with explained examples of custom type definitions.
- Support selectable color themes. I.e., change many faces by setting one variable.
- Support undo using lmi's '`gnus-undo.el`'.
- Make it possible to append to 'choice', 'radio', and 'set' options.
- Ask whether set or modified variables should be saved in `kill-buffer-hook`.

  Ditto for `kill-emacs-query-functions`.
- Command to check if there are any customization options that does not belong to an existing group.
- Optionally disable the point-cursor and instead highlight the selected item in XEmacs. This is like the *Completions* buffer in XEmacs. Suggested by Jens Lautenbacher '`<jens@lemming0.lem.uni-karlsruhe.de>`'.
- Explain why it is necessary that all choices have different default values.
- Add some direct support for meta variables, i.e. make it possible to specify that this variable should be reset when that variable is changed.
- Add tutorial.
- Describe the `:type` syntax in this manual.
- Find a place is this manual for the following text:

  Radio vs. Buttons

  Use a radio if you can't find a good way to describe the item in the choice menu text. I.e. it is better to use a radio if you expect the user would otherwise manually select each item from the choice menu in turn to see what it expands too.

  Avoid radios if some of the items expands to complex structures.

  I mostly use radios when most of the items are of type `function-item` or `variable-item`.
- Update customize buffers when `custom-set-variable` or `custom-save-customized` is called.
- Better handling of saved but uninitialized items.
- Detect when faces have been changed outside customize.

- Enable mouse help in Emacs by default.
- Add an easy way to display the standard settings when an item is modified.
- See if it is feasible to scan files for customization information instead of loading them,
- Add hint message when user push a non-pushable tag.

  Suggest that the user unhide if hidden, and edit the value directly otherwise.
- Use checkboxes and radio buttons in the state menus.
- Add option to hide '`[hide]`' for short options. Default, on.
- Add option to hide '`[state]`' for options with their standard settings.
- There should be a way to specify site defaults for user options.
- There should be more buffer styles. The default 'nested style, the old 'outline' style, a 'numeric' style with numbers instead of stars, an 'empty' style with just the group name, and 'compact' with only one line per item.
- Newline and tab should be displayed as '`^J`' and '`^I`' in the `regexp` and `file` widgets. I think this can be done in XEmacs by adding a display table to the face.
- Use glyphs to draw the `customize-browse` tree.

  Add echo and balloon help. You should be able to read the documentation simply by moving the mouse pointer above the name.

  Add parent links.

  Add colors.

Table of Contents