The Monitoring Grapical User Interface editor

Marc W. Mengel
Fermilab User Application Support Group

May 30, 1997

Contents

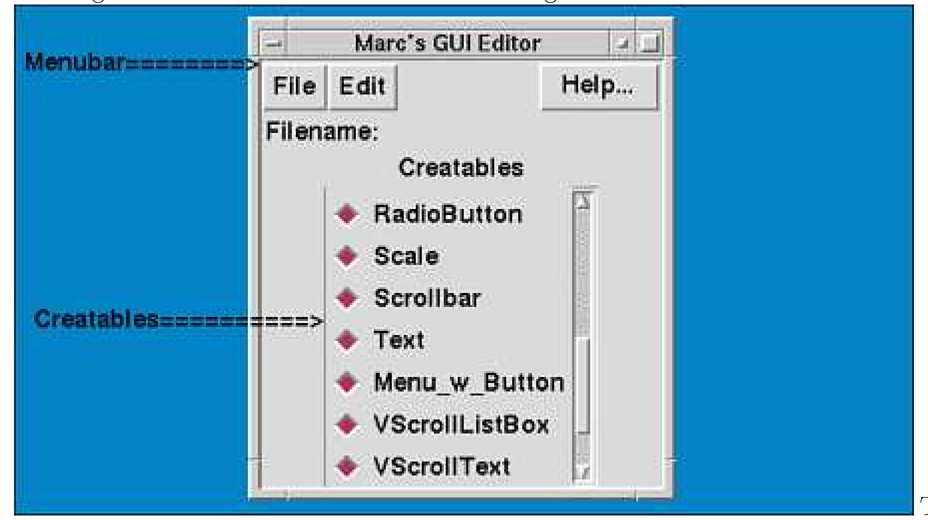
- Tutorial Introduction (cf. Section 2) to mgui
- Help Screen (cf. Section 1) for mgui
- Advanced (cf. Section 4) usage of mgui
- Importing (cf. Section 3) interfaces from Tcl scripts.

1 The mgui Windows

[see here for an all-ascii version]

1.1 Main Window

The mgui main window is divided into two regions:



The Menubar area has the File menu, the Edit menu, and the Help button, which provides this popup. The File menu has items to let you save and load interfaces, and to load new Composites into the Creatable area. The Creatables section lists names of widgets or groups of widgets which can be created with mgui.

1.2 Widget Tree Window

The Widget Tree lists all the windows currenty under mgui's control. You click on the items in the Widget tree list to get a menu of things to do to the listed widget.

1.2.1 Creating Widgets

To create a widget, click on its name in the Creatables window, and then click on the managed window in which you want to place the widget. Note that for Toplevel widgets, no second click is required unless your Window Manager requires it, since Toplevel widgets are not placed inside any other widgets. Note that doing the second click of a widget inside of another widget will make it that widget's child. Mgui will let you place buttons inside buttons, etc. so be careful where you make the placement click.

1.2.2 Moving/Resizing Widgets

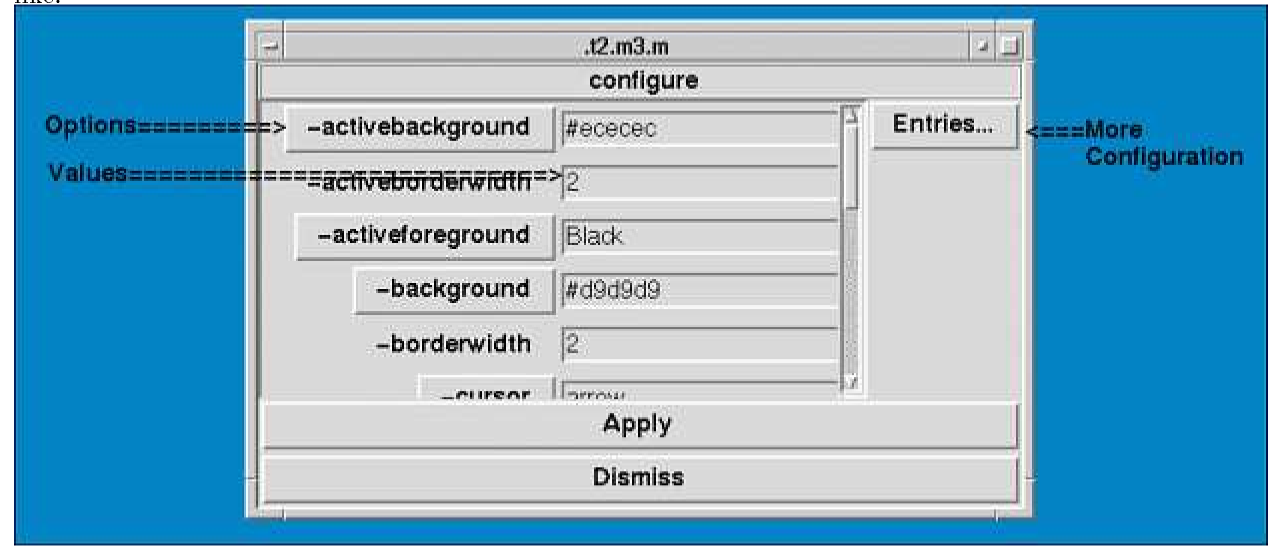
There are several ways to move and resize widgets. Note that we refer to moving only within the same parent window. Moving a widget from one parent to another is referred to as "reparenting", which is described later. One is way to move a window within its parent is by dragging the widget with mouse button 2 and 3 to resize and move it (respectively). Adding user-created bindings for these events will override the mgui bindings. However, even if you have overridden all of mgui's bindings, you can still get the menu from the Widget tree for the window, and use the Geometry popup to move or arrange the widget.

1.2.3 Configuring/Changing Widgets

Configuring widgets can be done via the widget's menu popup, which is done by:

- Shift-Mouse-Button-1 on the widget (that is holding down the shift key on the keyboard while pressing mouse button 1 on the widget.)
- Clicking on the Widget Tree listing for the widget.

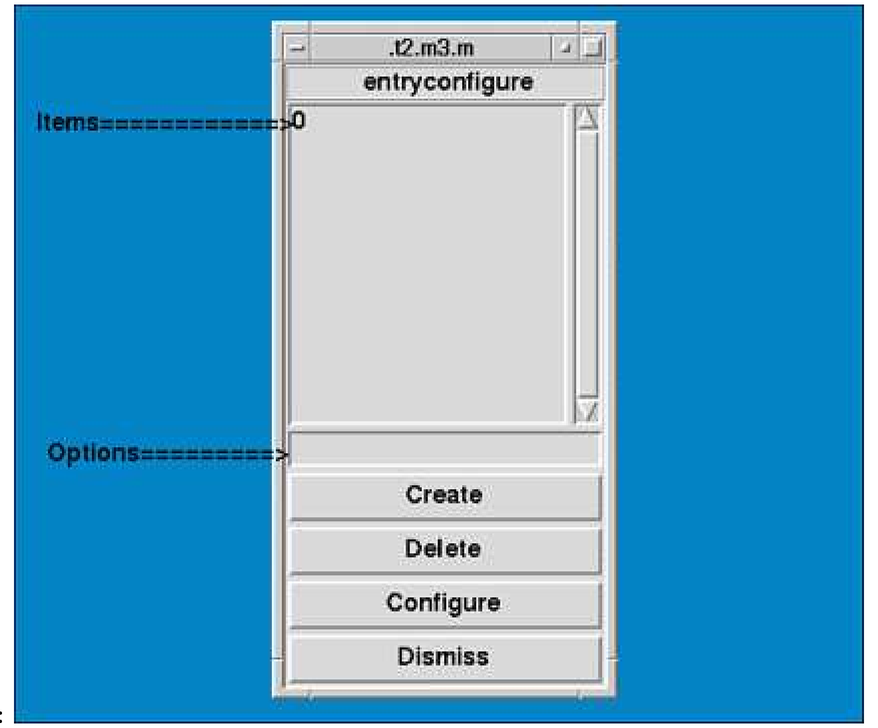
Either of these will pop up a menu. If you want to get rid of this menu without doing any of the listed items, simply move your mouse cursor out of the popup. You can get at the widget's configuration options by Configure from this menu. This will get you the basic configuration window for that widget, which looks like:



This lets you set options like foreground and background color, titles, commands, etc. The scroll bar lets you reach options further down the list. Some options are on raised buttons, which means that if you press them you get a menu of values to choose from. picking an item from that menu sets the option immediately. Typing a value in an entry doesn't set it until you either press in the entry, or click on the Apply button. Clicking on Dismiss makes the whole window go away, and forgets any not-yet-Applied changes. The buttons in the upper right hand corner are further configuraion windows, which either look like this one, or pop up an item selection window.

1.2.4 Item selection windows:

Item selection windows look like:



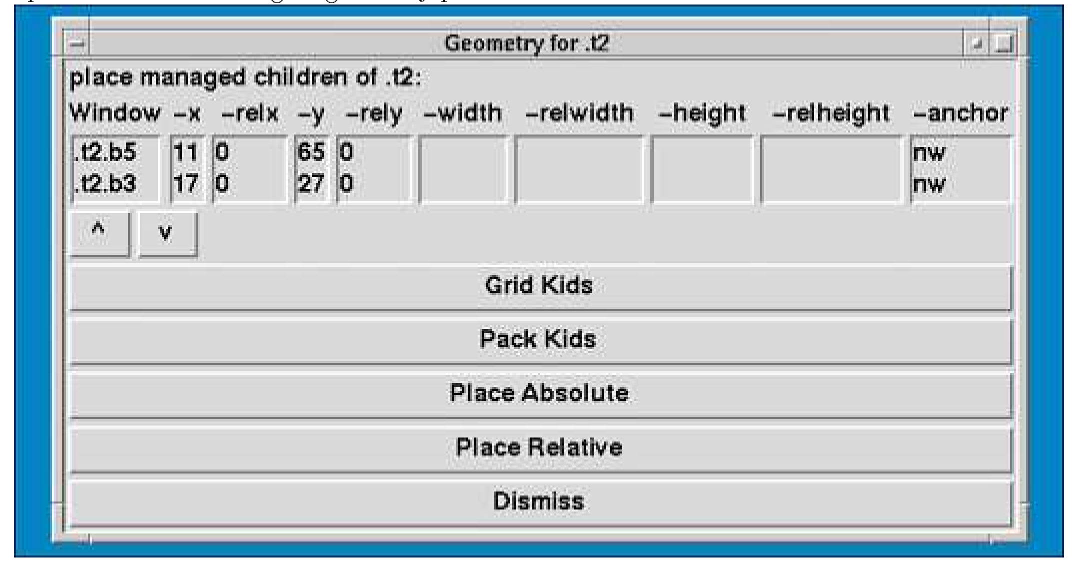
They are used to configure widget entities that have "items" which can be created, destroyed, etc. For example, Menus have menu items, Text widgets have tags, etc. all of which can be created or modified using this window. If any "items" already exist, they are listed in the Items listbox. Selecting one and clicking on Configure will pop up a configure window for that item. Selecting one and clicking on Delete will remove that item (if possible). Clicking on create will try to create an item, using the options in the entry directly above the button. For example if you are configuring menu items, you need to put the type of item ("command" or "tearoff" or ...) and any other initial options in the entry, and then press Create.

1.2.5 Geometry Management

How windows are arranged within their parents is the subject of the geometry management window for a given widget. You can reach it by:

- Shift-Mouse-Button-1 on the widget
- Clicking on the Widget Tree entry for the widget

Either of these will pop up a menu. If you want to get rid of this menu without doing any of the listed items, simply move your mouse cursor out of the popup. You can get at the widget's Geometry management by picking Geometry. This will get you the Geometry management widget for the window. This gives you a table of all of the managed children of the widget, and tables to set various options about the widgets geometry placement.



There are up [^] and down [v] buttons below the list of windows, which allow you to rearrange the order of the windows in the list. (This is especially important for packed widgets). If you click on a child's widget name, and then on the up [^] or down [v] button, it will be moved up or down (respectively) in the list, and the window will be redrawn.

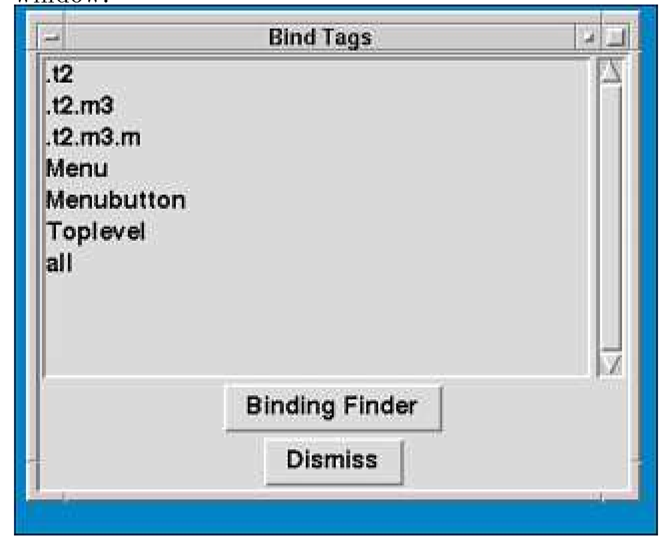
Specific values in the table can be changed by clicking on them, which will create an entry widget on top of the item, where you can change it, and press [Enter] to make the change.

There are also several buttons at the bottom of the window, which allow you to pack all the child widgets, place all the child widgets with absolute coordinates, or place all the child widgets with relative coordinates. All of these will perform the task then redraw the window.

Finally there is a Dimsiss button, which will make the Geometry window go away.

1.2.6 Binding Management

In the Edit menu there is a Bindings window which lists the various bindtags currently in use. Note that if you haven't created any widgets yet, only "all" will be listed here. After widgets are created, bindtags for both the widget's class and for its particular widget name will be listed when you pop up this window.



Clicking on a bindtag will give you another popup with a list of its currently bound events.

In this window you can click on existing event to modify the command it is bound to, or you can enter an event into the entry at the bottom of the screen, and press the [Add] key to create a new event binding, and get a screen to edit that binding. The event command editor screen looks like:

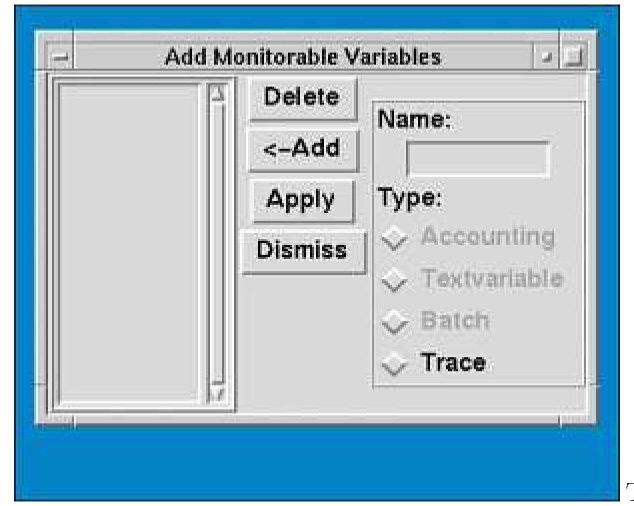
and provides rudimentary text editing. As usual, Apply changes the actual bind command, while Dismiss will get rid of the window without making any un-Applied changes.

The [Event Finder] button on the prevous screen pops up 2 windows. One is a "victim" window, to which things can be done. The other is an event log window, which logs all the events received by the victim. You can cut and pased from the event log into the entry next to the Add button, then press the Add button to set the command for the newly found and added event.

1.2.7 Monitoring Variables

There are two components to monitoring; the first is the list of monitor-able variables, the second is setting up individual widgets to be monitor variables. This becomes useful mainly with the addition of the BLT widgets, particularly bar charts, but you can monitor variables' values with label widgets, for example.

To add a variable to the list of monitorable variables, you pull down the Edit menu, and choose [Monitorable...]. This pops up a window with a list of existing variables and types on the left; some operator buttons, and a section where you can specify new variables to add to the list.



To add a new variable, type its name in the entry field at the upper right, click on a monitor type (currently only Trace is supported) and hit the [As usuall, hit [Apply] to have this happen, and [Dismiss] to make the window go away, ignoring any un-Applied changes.

Managing the monitoring duties of individual widgets is the job of the monitoring management window for a given widget. You can reach it by:

- Shift-Mouse-Button-1 on the widget
- Clicking on the Widget Tree entry for the widget

Either of these will pop up a menu. If you want to get rid of this menu without doing any of the listed items, simply move your mouse cursor out of the popup. You can get at the widget's Monitoring management by picking Monitoring. This will get you the monitor widget, which lists the currently monitored varibles on the right, and the monitorable widgets on the left.

2 An Introduction to mgui – Creating a simple text editor

Building a simple application with the "mgui" user interface develpment tool is relatively simple. This document will take you through building a simple text editor with "mgui", and then describe how to save it as a standalone application, or as a "composite" which can be loaded into mgui, or used as a subroutine in other Tcl/Tk scripts. Next it will describe how to use mgui as an addition to other Tcl/Tk applications. Finally, it explains how to add items to "mgui"'s startup "Creatable" list, so that new composites, or widgets from toolkits "mgui" doesn't yet know about can be added.

2.1 Outline

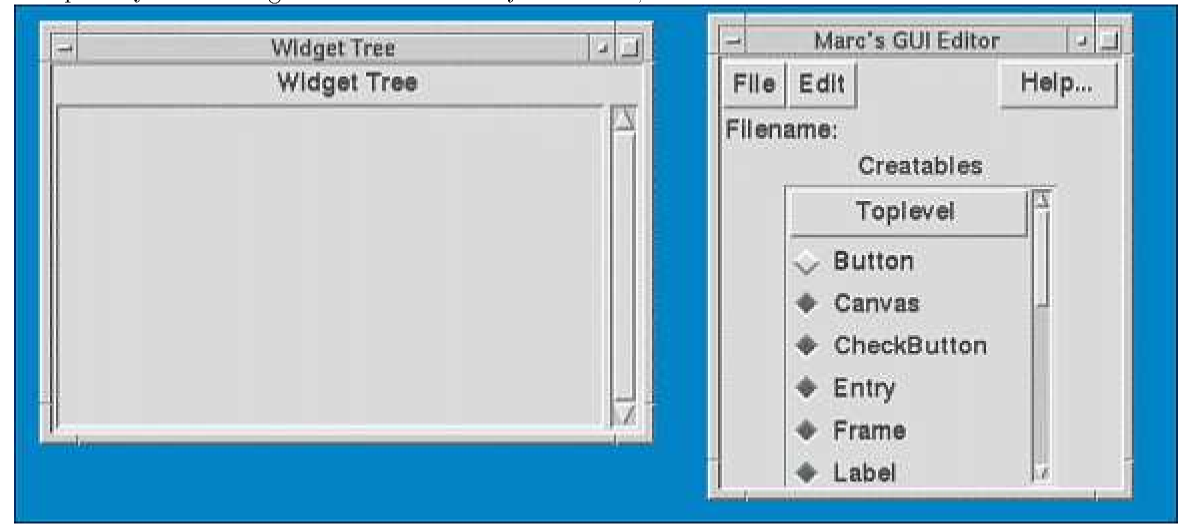
- Starting mgui (cf. Section 2)
- Creating the Basic Widget Tree (cf. Section 2.3)
- Making Load work (cf. Section 2.3.2)
- Making Save work (cf. Section 2.3.3)
- Saving the Application (cf. Section 2.3.4)
- Making Resizing work (cf. Section 2.3.5)
- Saving the Application as a Composite (cf. Section 2.3.6)

2.2 Starting "mgui"

To start mgui, you can:

- run "mgui" as a shell script, making sure the directory "mgui" lives in is in your command search path. For a Tcl/Tk command interperter it will use the current definition of the environment variable `GUIBUILDER`, defaulting to `wish4.2`
- set the environment variable `MGUI_DIR` to point to the directory where "mgui" is installed, and run `wish -f $MGUI_DIR/mgui`
- set the environment variable `MGUI_DIR` to point to the directory where "mgui" is installed, and run a Tcl/Tk interpreter interactively, and `source "$env(MGUI_DIR)/mgui"`

At this point you should get two windows on your screen, like



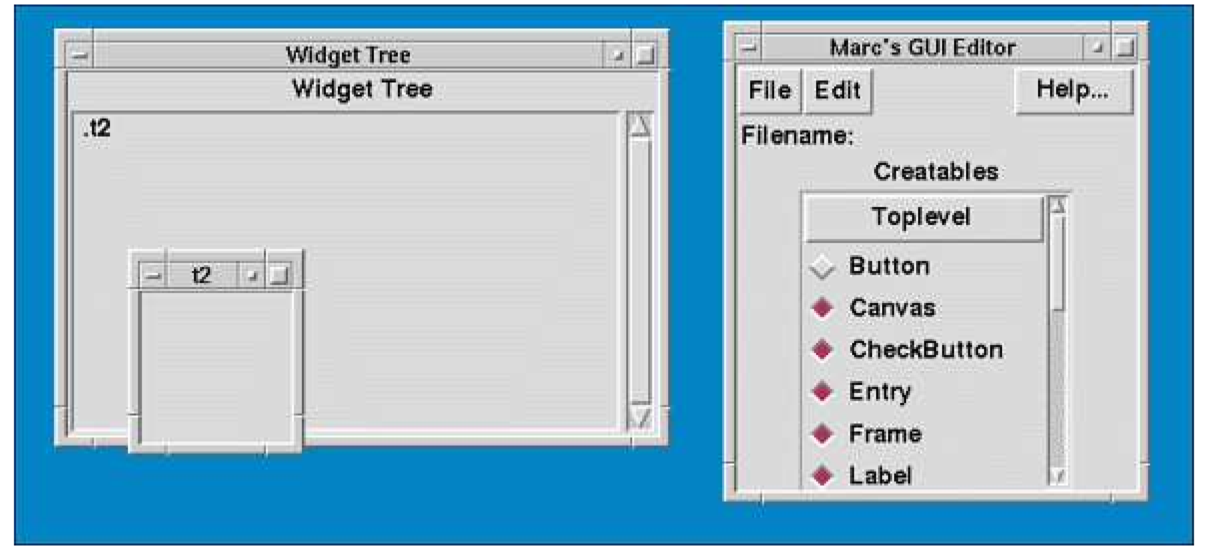
Next (cf. Section 2.3)Building the Widget Hierarchy

2.3 Building the Widget Hierarchy

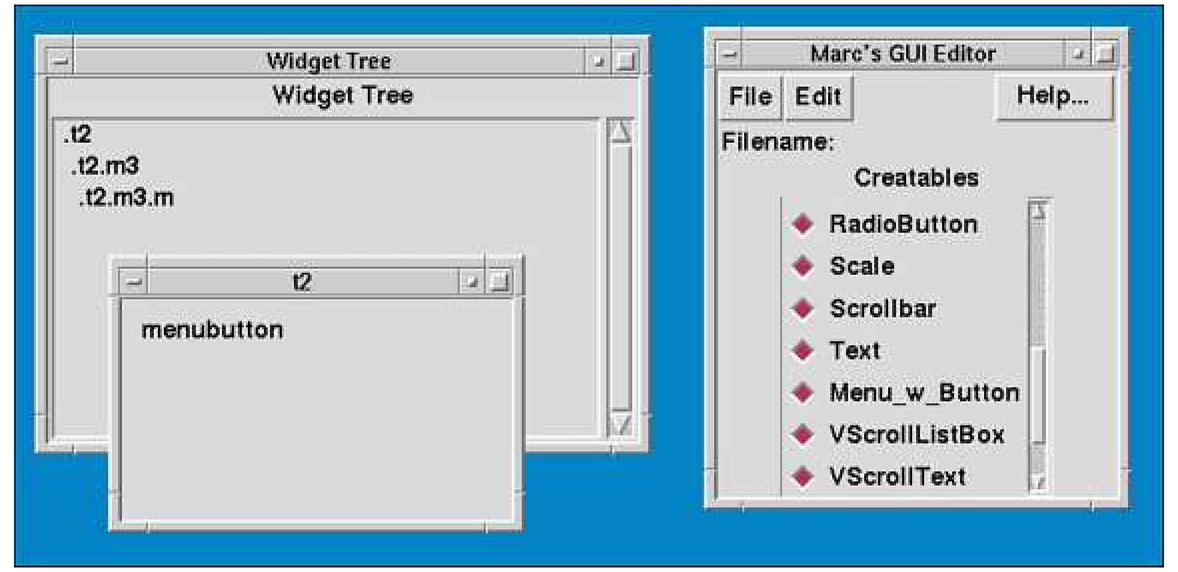
Up (cf. Section) to mgui Help Overview Next (cf. Section 2.3.2) Making Save Work Prev (cf. Section 2) Starting mgui

---

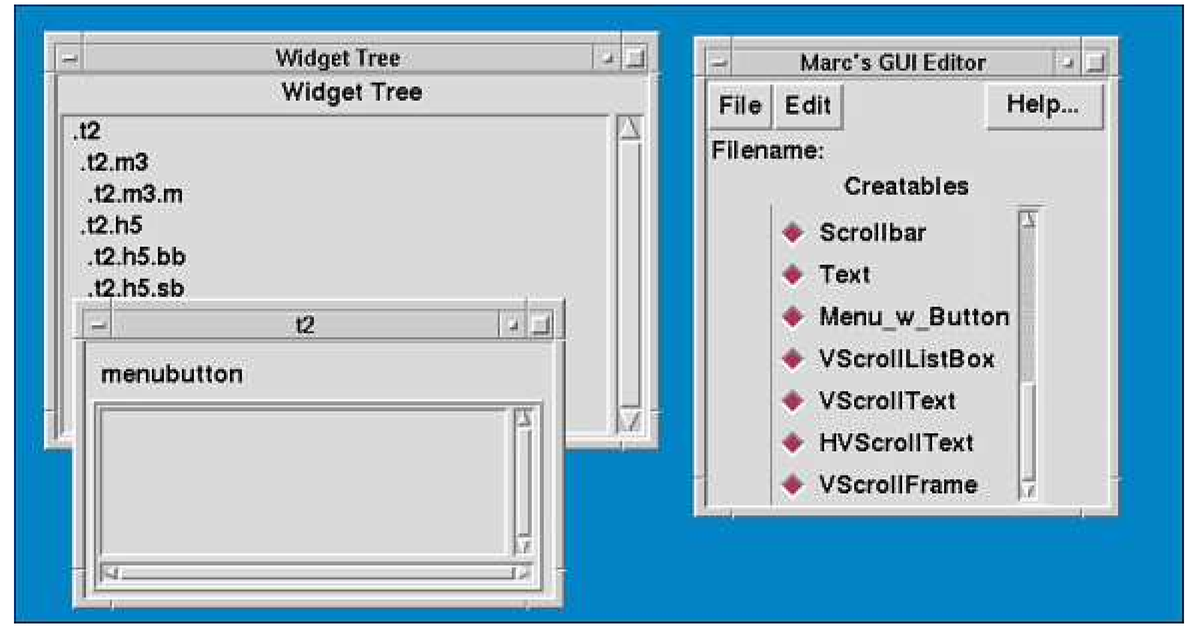
To build our text editor, we first need a toplevel widget to hold our application. We create one by pressing on the button labeled **Toplevel** in the Creatables window. This will make a new toplevel widget named ".t2", which will show up on the screen, and whose name will appear in the Widget Tree window, as in



Next we'll add a File menubutton and menu. We'll start off by stetching our toplevel window a bit, so we have some room, and then click on the button labelled **Menu_w_Button** in the Creatables Window (you have to scroll down the Creatables scrollbar to do this). Then our cursor turns into a button-bitmap, and we can bring the mouse cursor into the toplevel window, and click on the toplevel window to place the button, which looks like:



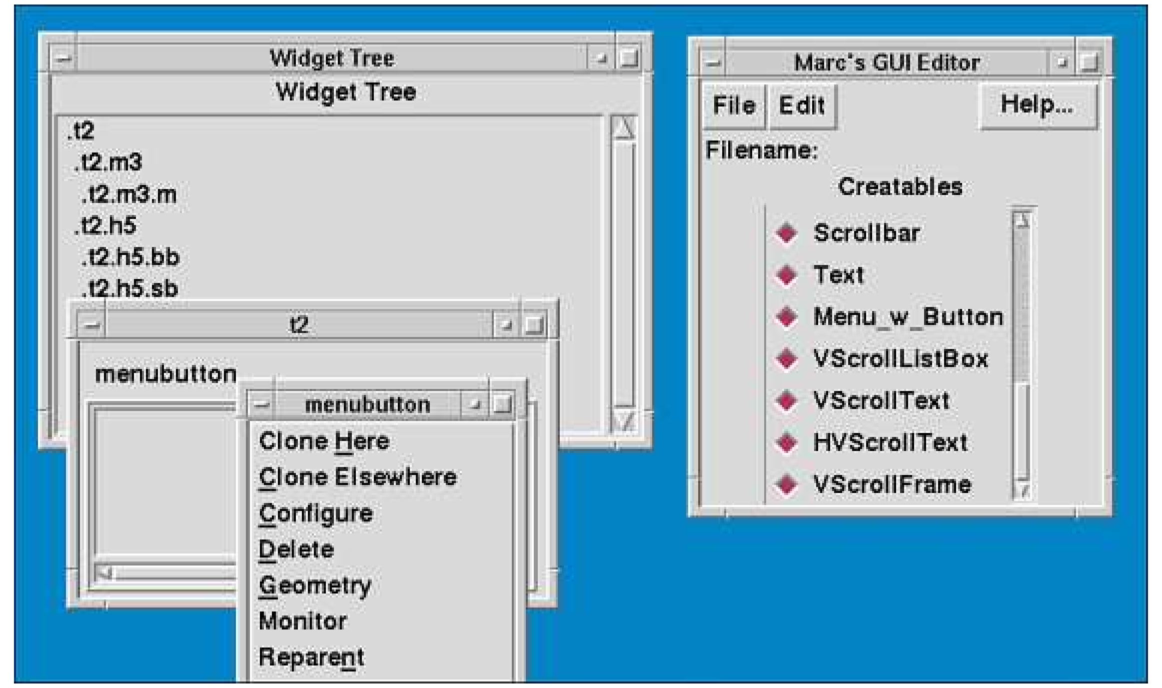
We can add a scrolled text area with the **HVScrollText** button the same way. Now we can make the scrolled text area larger by dragging the lower right hand corner with mouse button 2, and move it around by dragging the upper left hand corner with mouse button 3. This looks like:



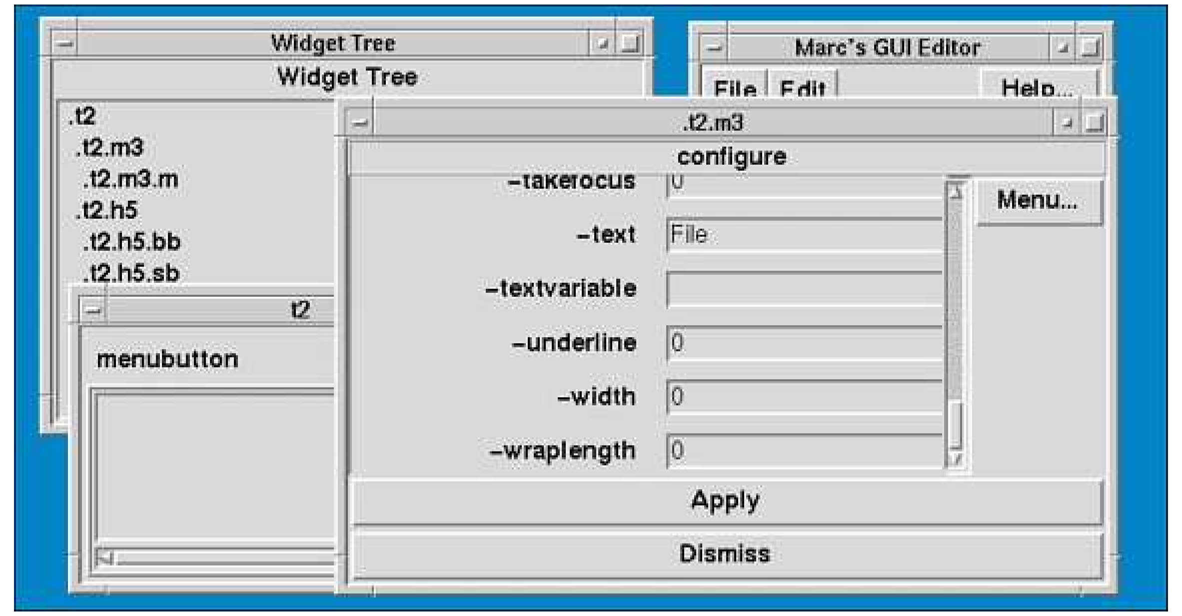
Next we need to do some configuration, to make our menu button say **File** on it, etc.

2.3.1 Configuring the File menubutton

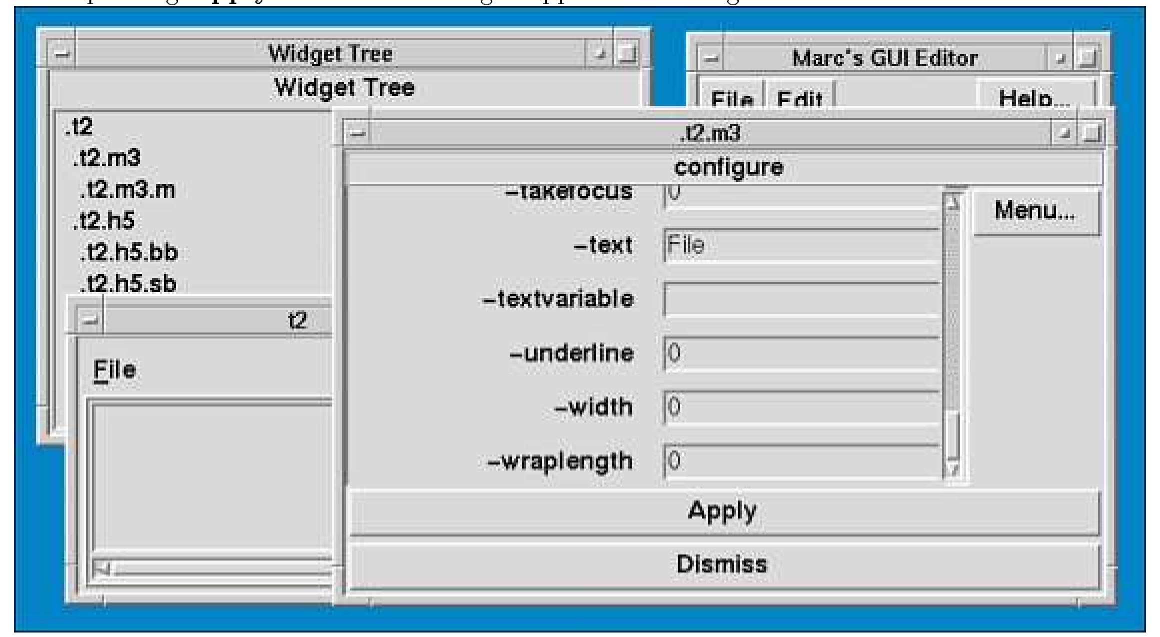
To configure our menubutton, we can hold down the shift key on the keyboard, and press mouse button 1 on our menubutton, which will pop up a menu:



and then select **Configure**. This will pop up the configure window for our menubutton. For the moment, we need merely to change the **-text** value by scrolling down on the scroll bar to the bottom and changing the "menubutton" in the entry next to **-text** to read "File", changing the **-underline** value to "0"



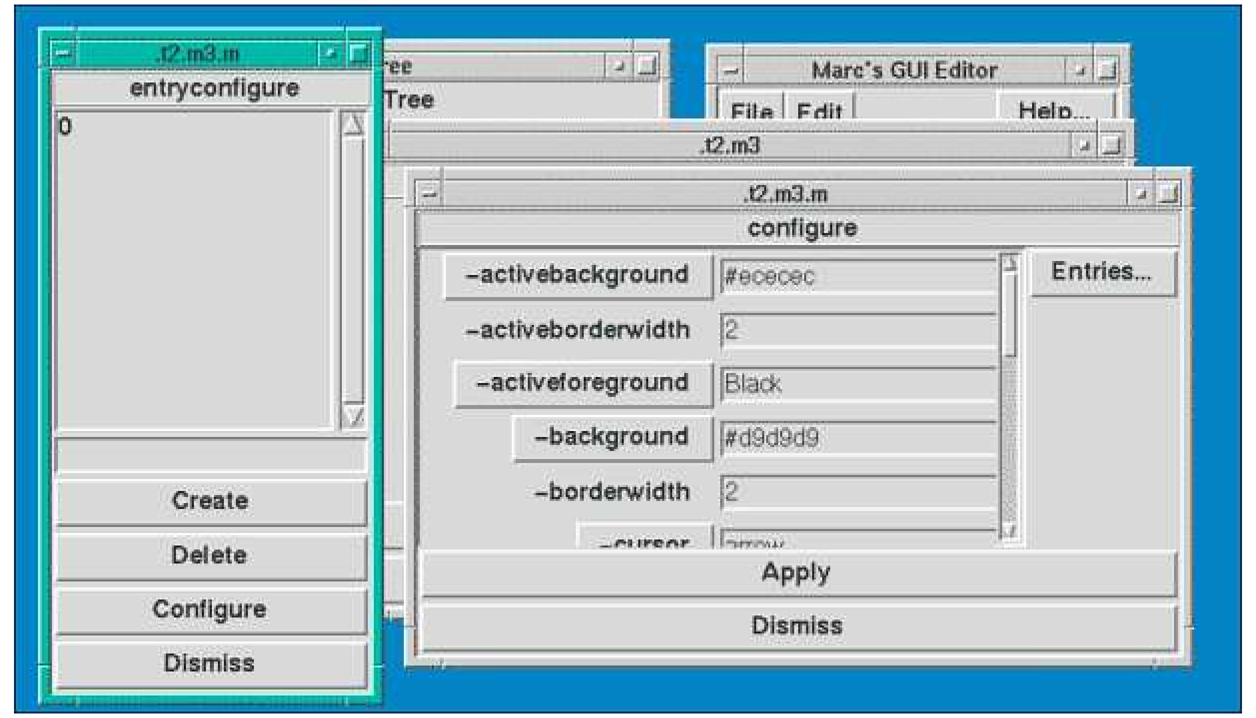
and then pressing **Apply** to make the change happen in the widget.



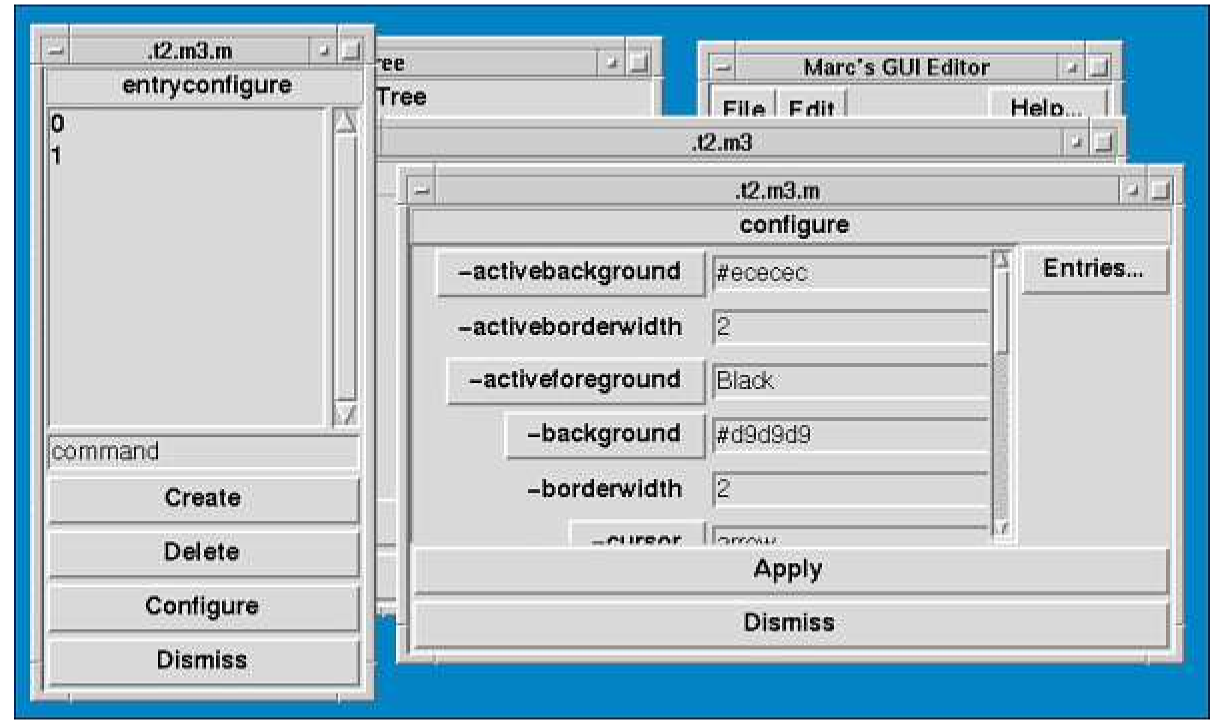
Next (cf. Section 2.3.2) Making Save Work Prev (cf. Section 2) Starting mgui

2.3.2 Adding a Load menu item

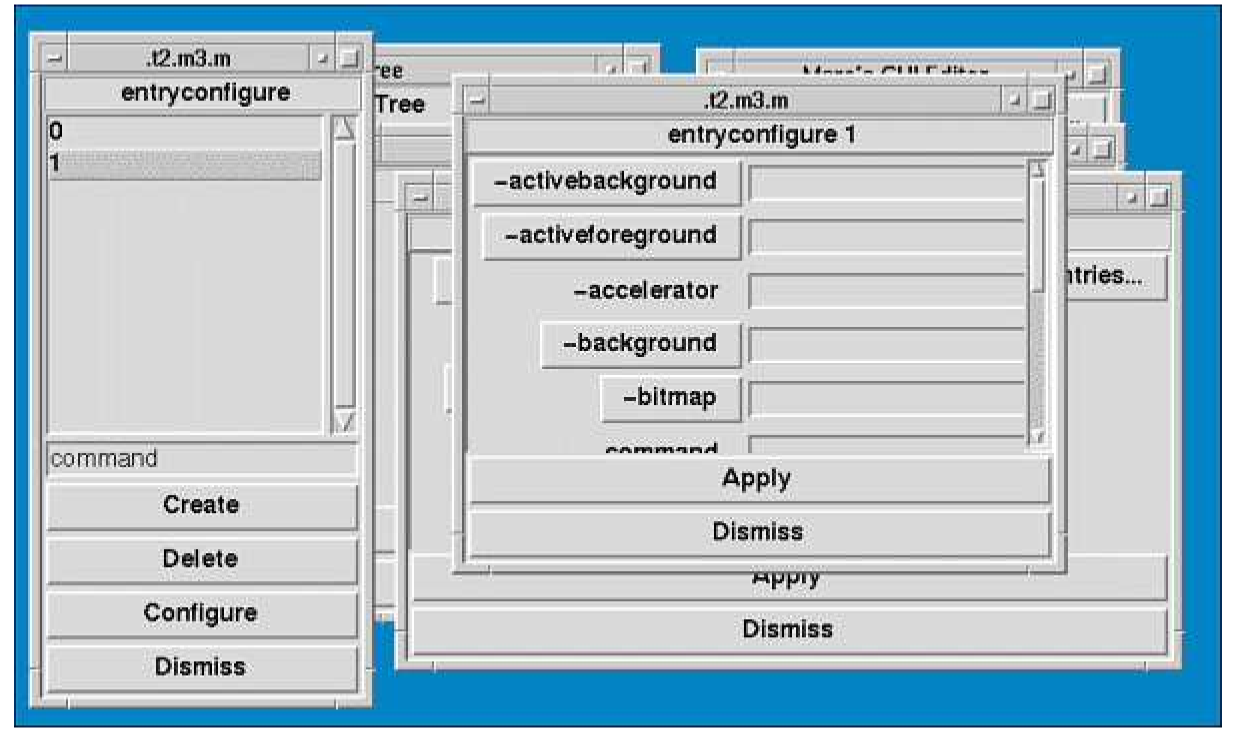
To configure the actual menu that pops up when we work on the menubutton, we can click on the **Menu...** button in our configure window. This lets us configure the basic menu attributes, but we want to add and configure new menu entries, so we need to click on the **Entries...** button which gets us the **entryconfigure** window for the menu.



From here we can **Configure** menu entries, **Create** them, and **Delete** them. We need to tell it which type of menu entry in the text entry box in the middle. To make a "Load..." file menu item we want a "command" menu entry, so we type "command" into the box, and hit **Create**



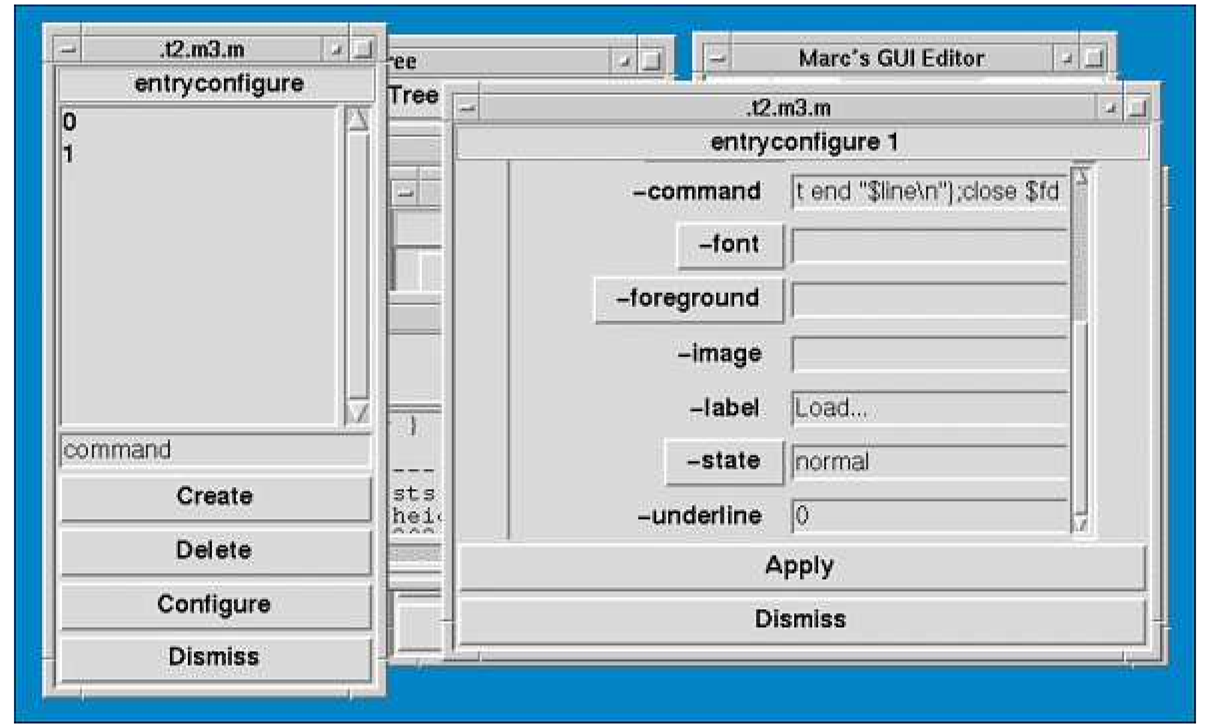
We can now configure this new item by clicking on the "1" and hitting the **Configure** button.



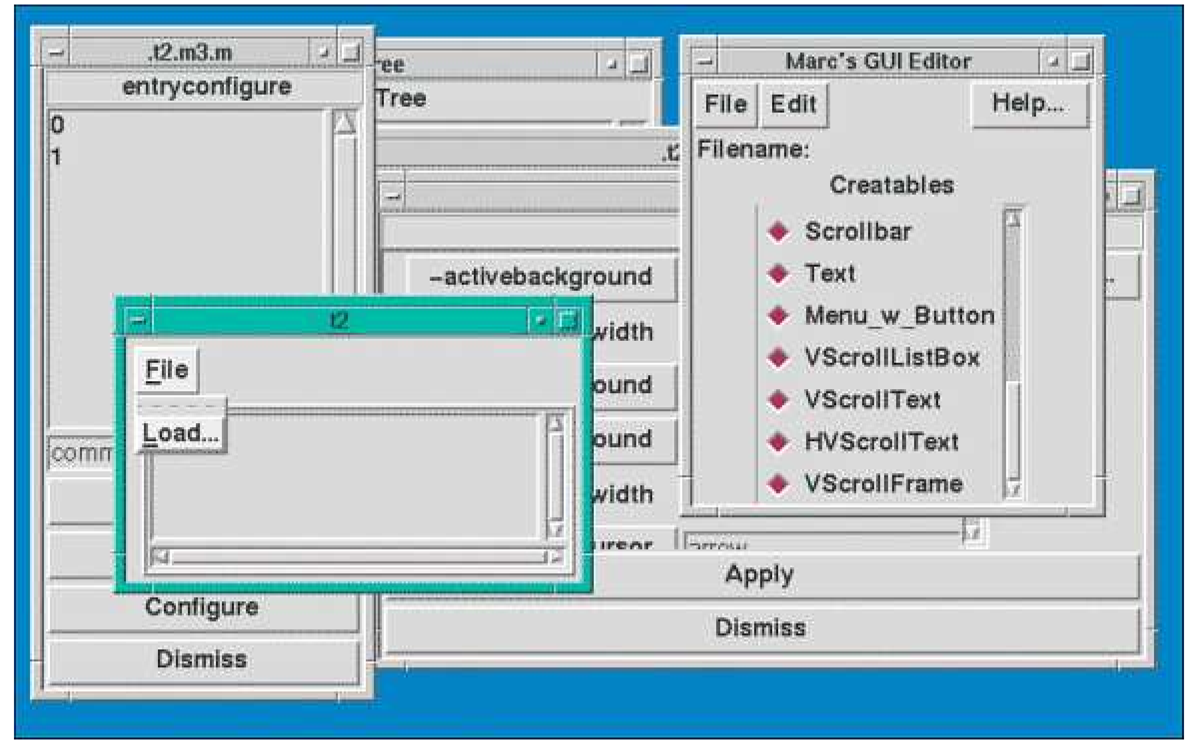
Our "Load..." menu item needs to have its **-label** changed to "Load...", its **-underline** changed to "0", and its **-command** changed to something to read in a file. The command I'm using (with linebreaks inserted for readability) is:

```
.t2.h4.t delete 0.0 end;
set fd [open [tk_getOpenFile] "r"];
while {0 <= [gets $fd line]} {.t2.h4.t insert end "$line\n"};
close $fd
```

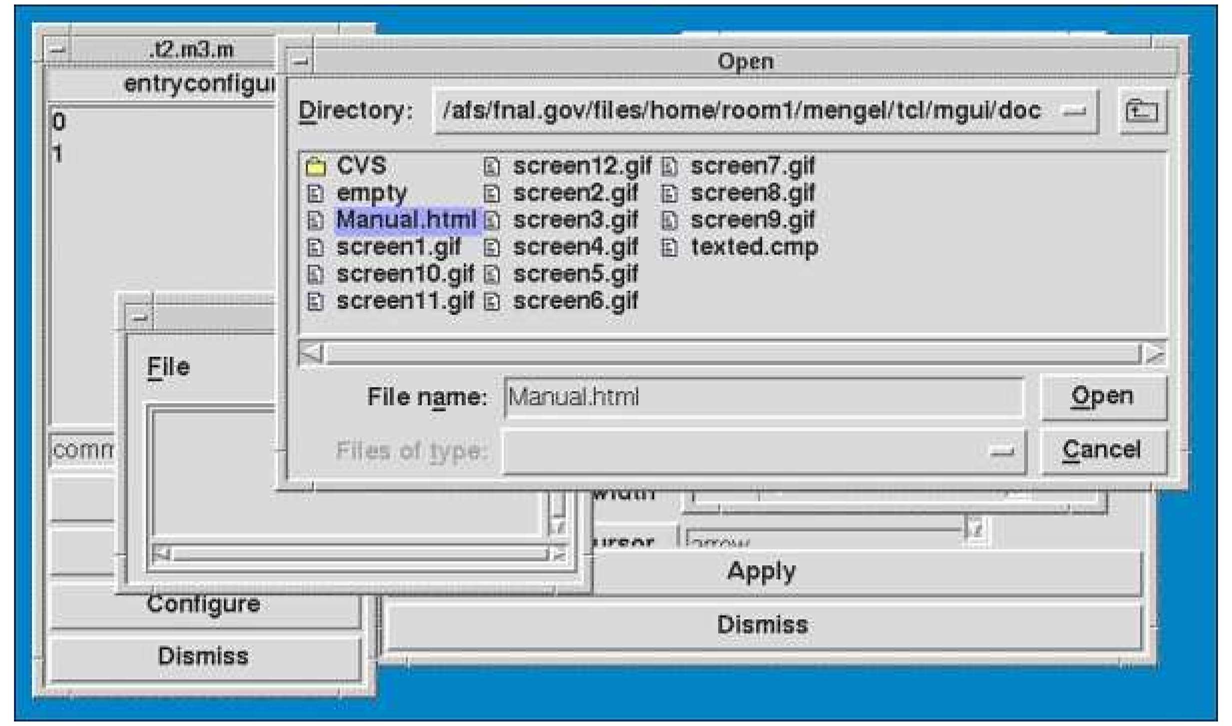
which deletes any existing text in the text widget **.t2.h5.t** (you can bring forward the **Widget Tree** window to remind yourself of the text widget's name); uses the tk_getOpenFile routine to pop up a filebrowser and get a filename from the user, opens that file, and reads each line from the file, and inserts it at the end of the text widget, and closes the file.



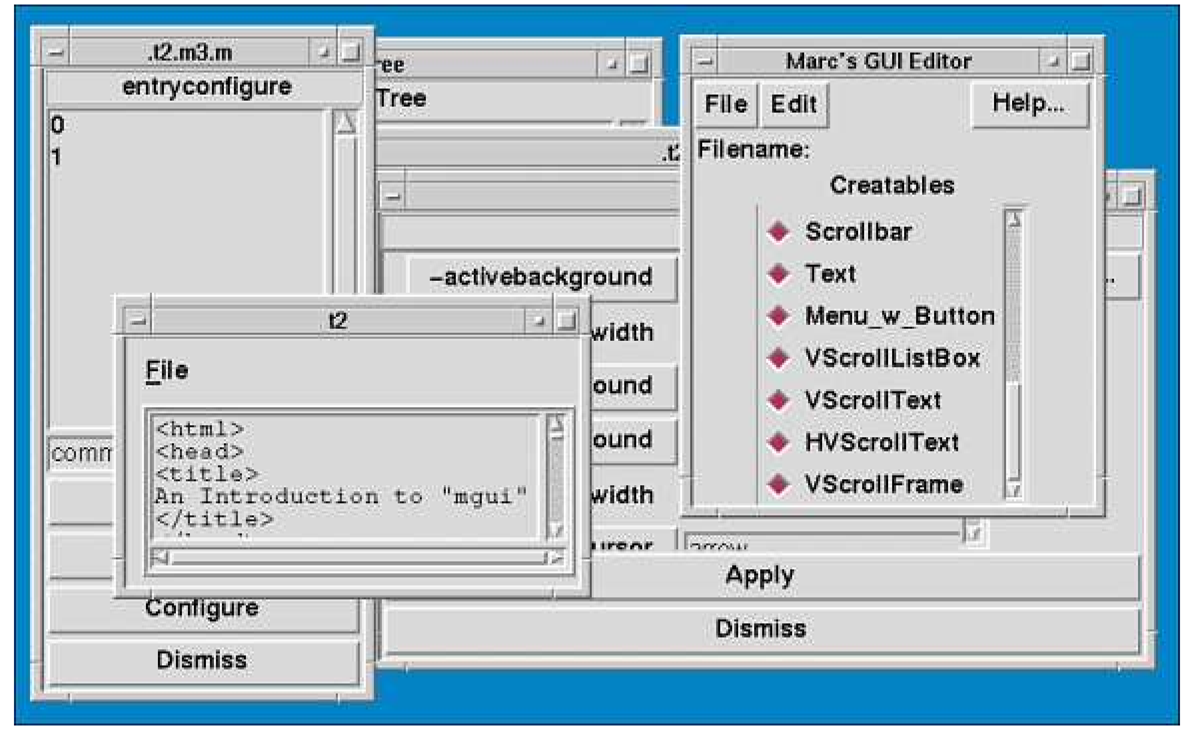
I can then **Apply** those changes, and **Dismiss** the **entryconfigure 1** window, and bring my application to the front. If I click on the **File** menu, I now see the **Load...** menu option.



and if I choose it, I get the Tk file browser,



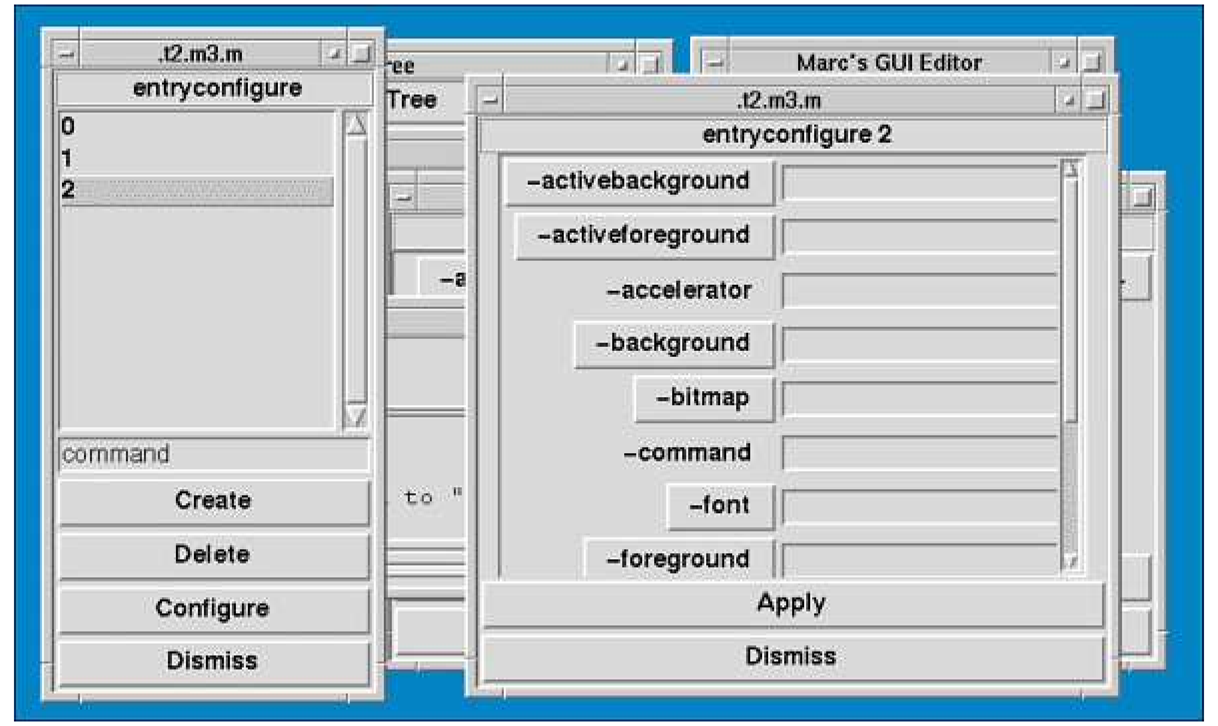
and can select a file and **Open** it.



Next (cf. Section 2.3.3) Making Save Work Prev (cf. Section 2.3) Building the Widget Hierarchy

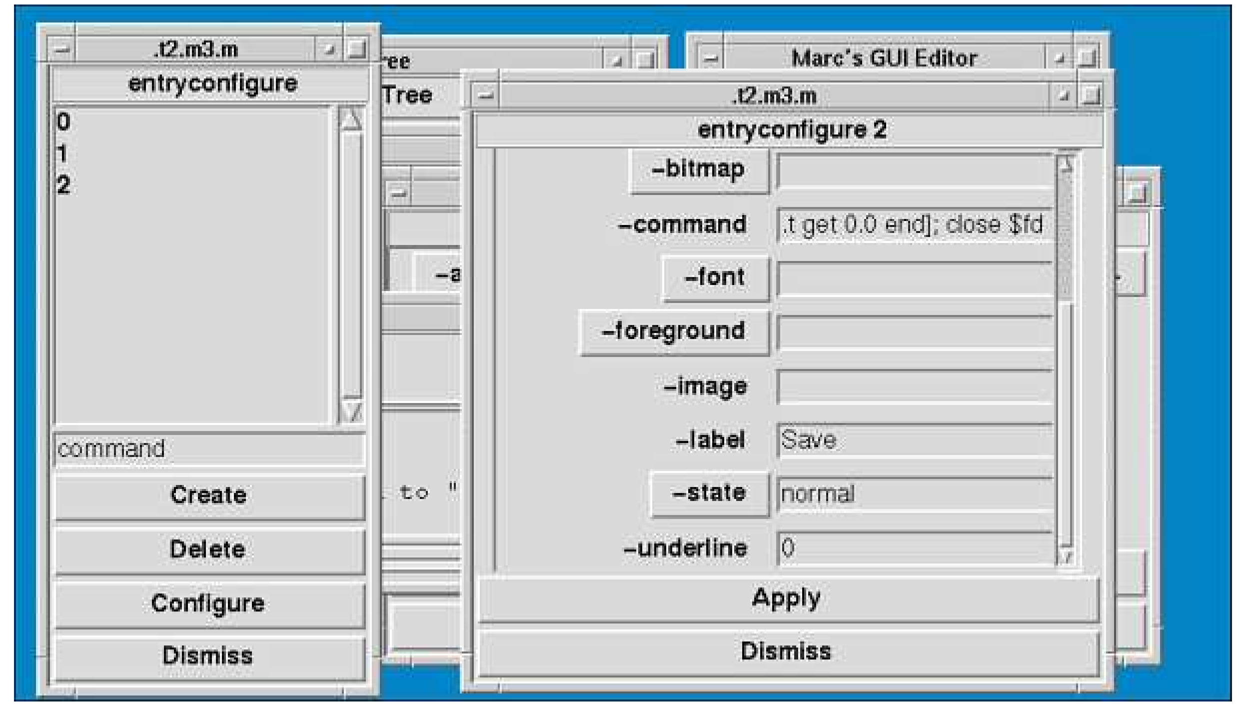
2.3.3 Adding a Save menu item

We can now go back to our **entryconfigure** window, and **Create** another "command" menu item, and **Configure** it.



This time we'll use a **-command** of:

```
set fd [open [tk_getSaveFile] "w"];
puts $fd [.t2.h4.t get 0.0 end];
close $fd
```



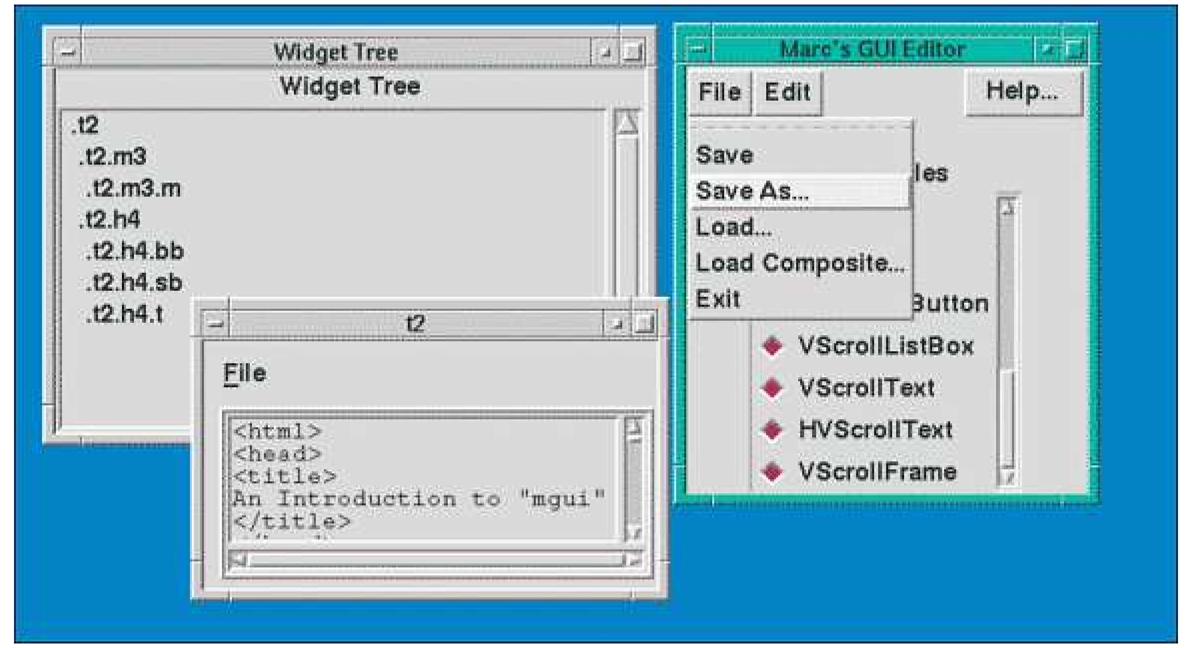
if we **Apply** those changes and our configure windows, we now have a working text editor.

Next (cf. Section 2.3.4) Saving our Application Prev (cf. Section 2.3.2) Making Load Work

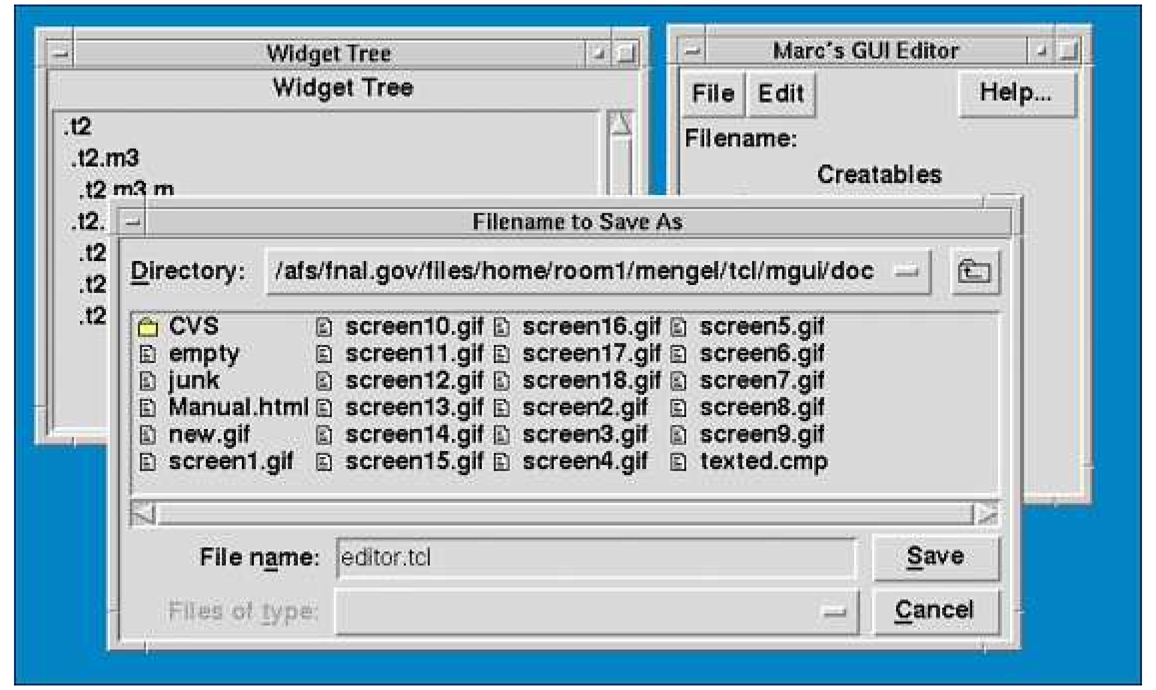
Up (cf. Section) to mgui Help Overview Next (cf. Section 2.3.5) Making Resizing Work Prev (cf. Section 2.3.3) Making Save Work

2.3.4 Saving our editor

We can write a Tcl/Tk script that is the editor we've built so far simply by choosing the **Save As...** menu item from the **File** menu in the main **Marc's GUI Editor** window,



filling in a filename (in this case "editor.tcl") in the file browser



and hitting **Save**

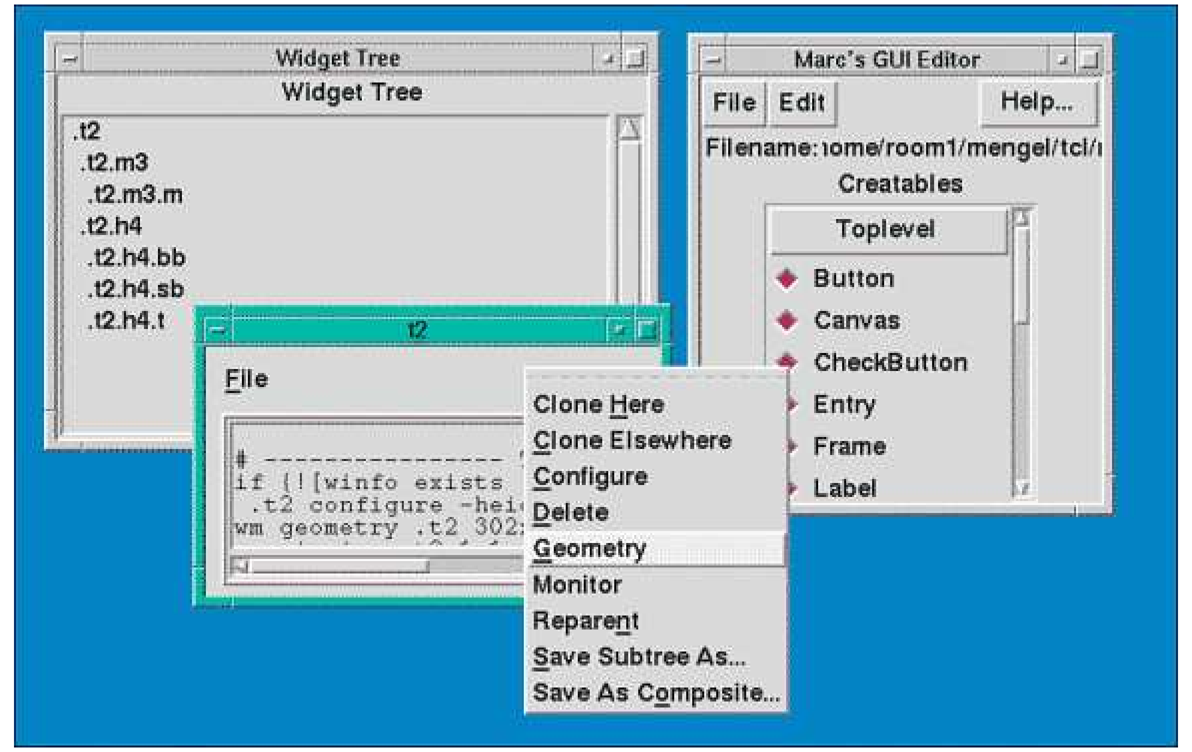
We can now use the editor to edit itself, and load our "editor.tcl" file into our text editor to look at it.

Next (cf. Section 2.3.5) Making Resizing Work Prev (cf. Section 2.3.3) Making Save Work

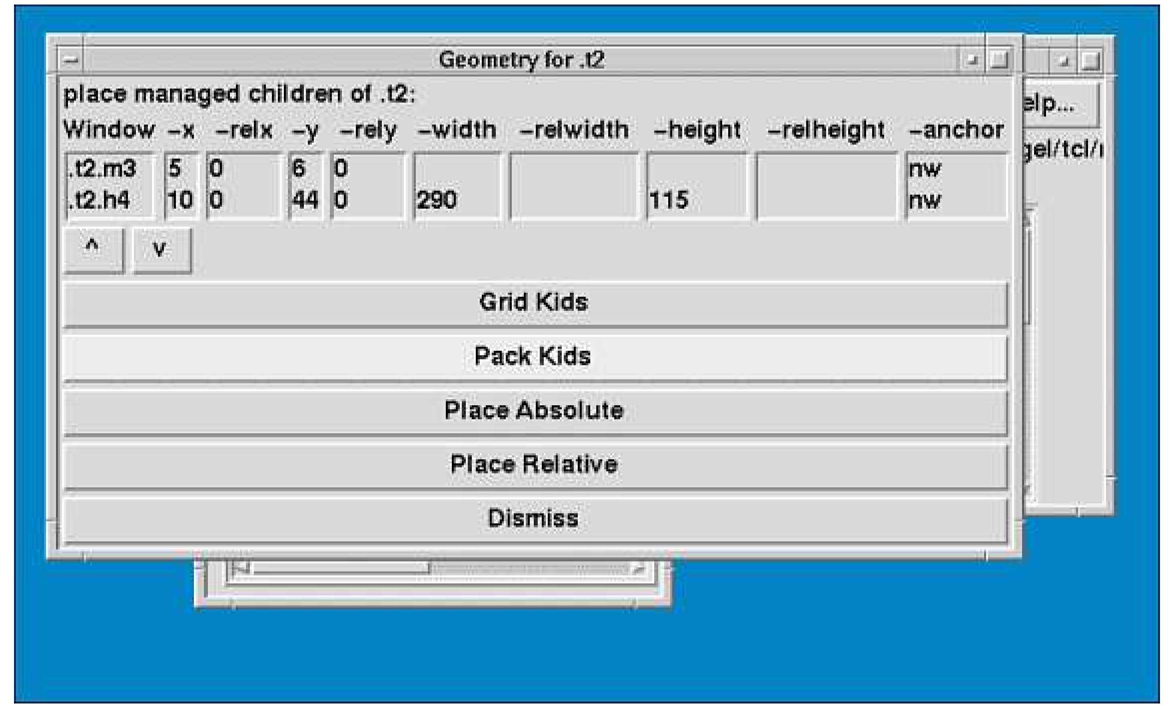
---

2.3.5 Making Resizing Work – Geometry Management

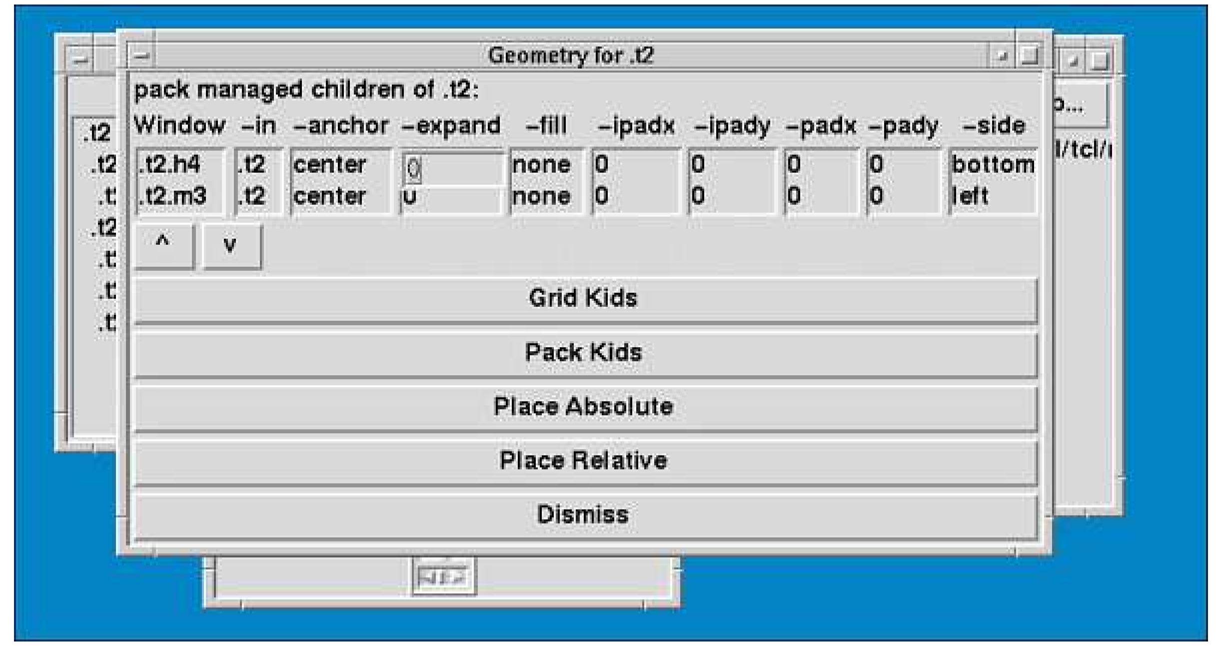
One drawback of our Tcl/Tk text editor so far is that if we resize the outer window, the text area doesn't stretch with it. To change this, we need to redo the geometry management of our window. We do this by holding down the shift key and pressing mouse button 1 on the t2 window to get our menu, and choosing **Geometry**



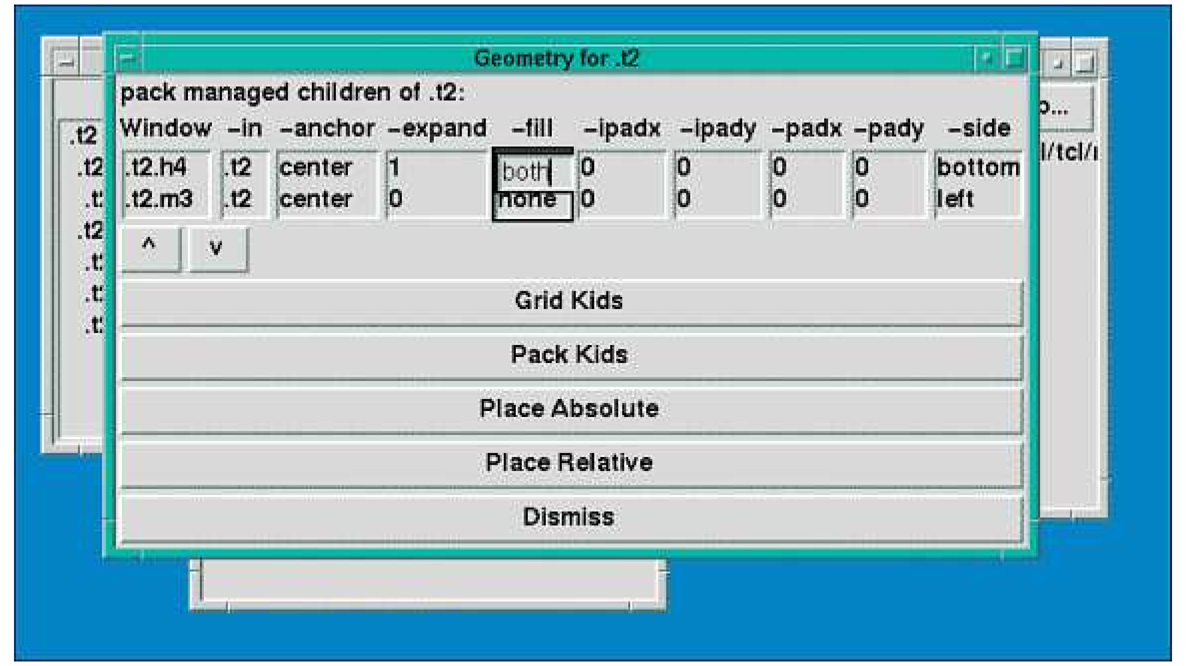
This screen then appears, showing that the widgets in the .t2 window are currently managed with "place", and have absolute x, y, width, and height attributes.



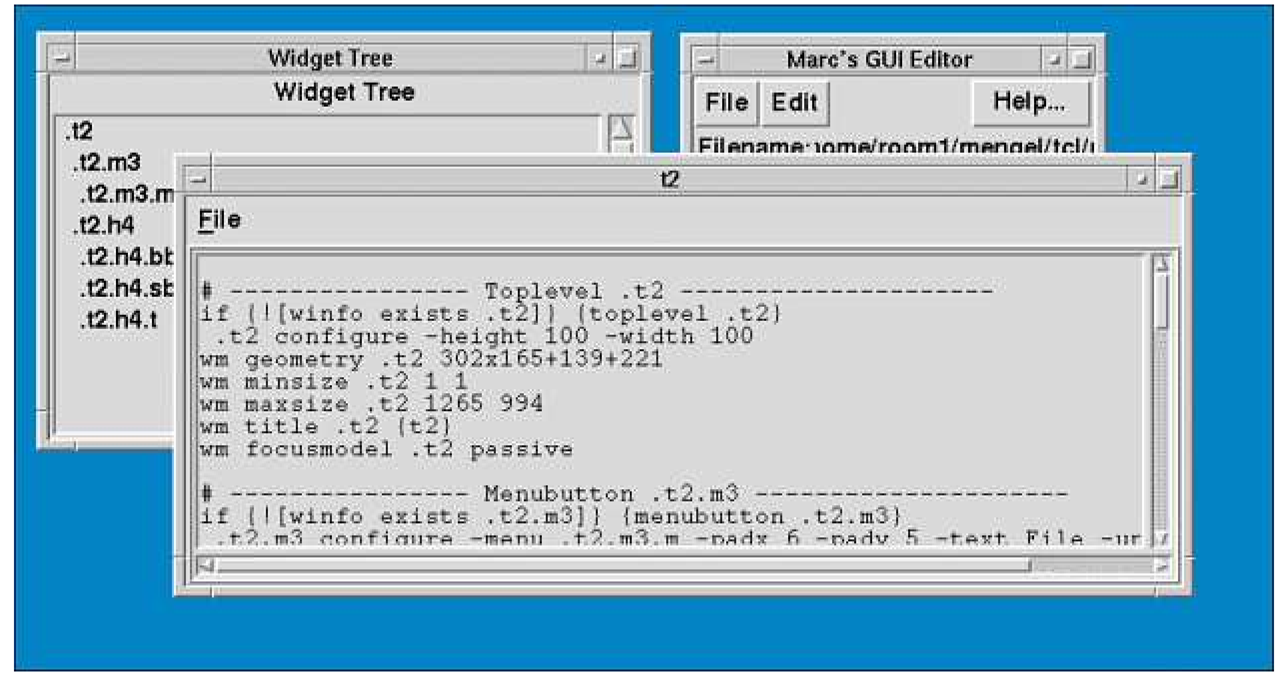
We can convert it to a "pack" based geometry management by using the **Pack Kids** button.



and then set the **-expand** option to "1" and **-fill** option to "both" by clicking on them and typing their new values in to the entry fields that appear, and thin pressing < Return > on the keyboard.



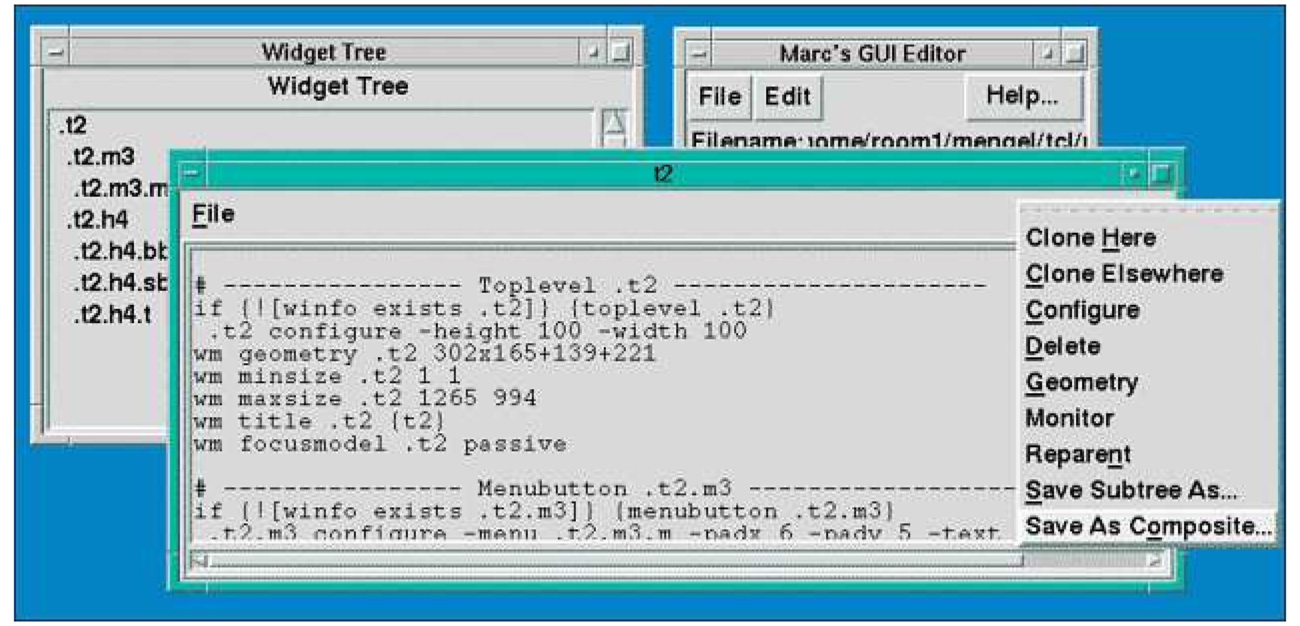
Finally, with our geometry options set properly, we can **Dismiss** the geometry window, and when we resize our text editor, the scrolled text region expands to fill the window.



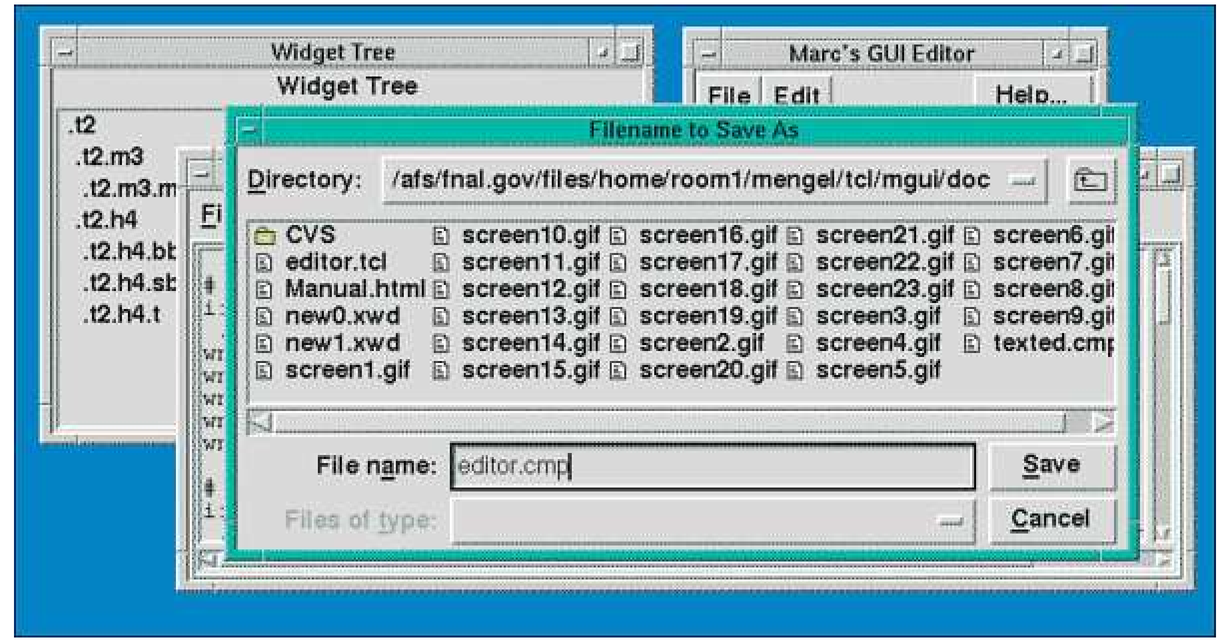
Prev (cf. Section 2.3.4) Saving our Application Next (cf. Section 2.3.6)Saving our Application as a Composite

2.3.6 Saving the editor as a composite

This is done by using Shift–button-1 to get the popup menu, and choosing **Save as Composite...**

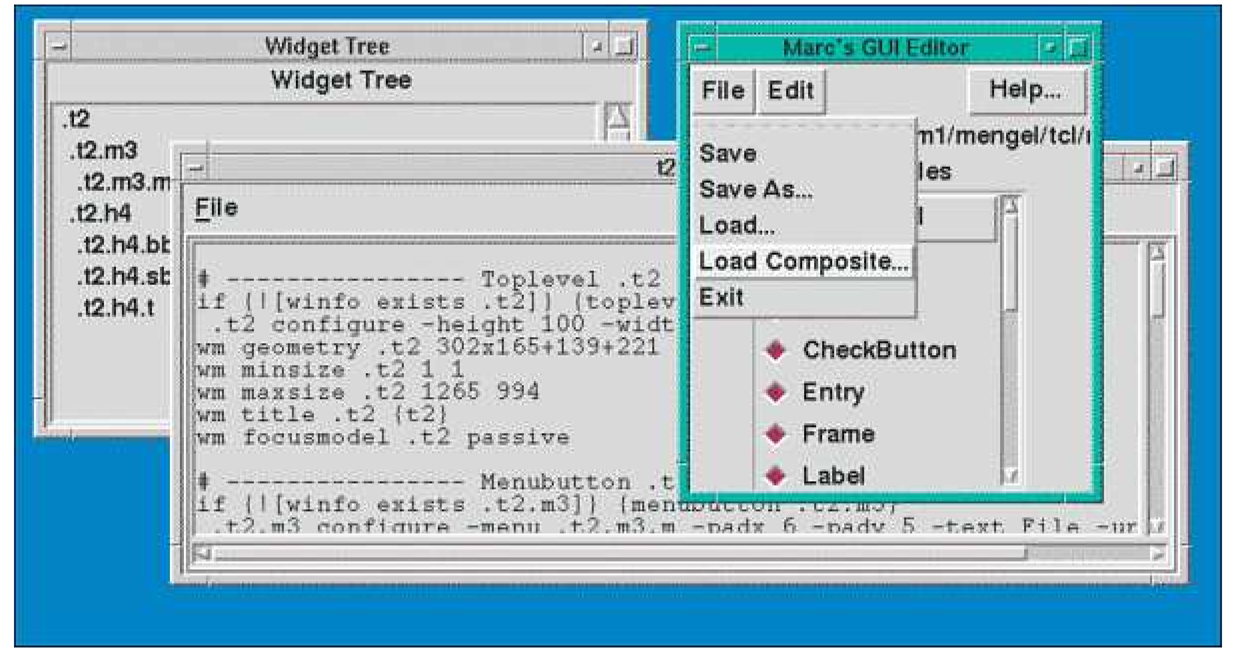


and filling in a filename and saving

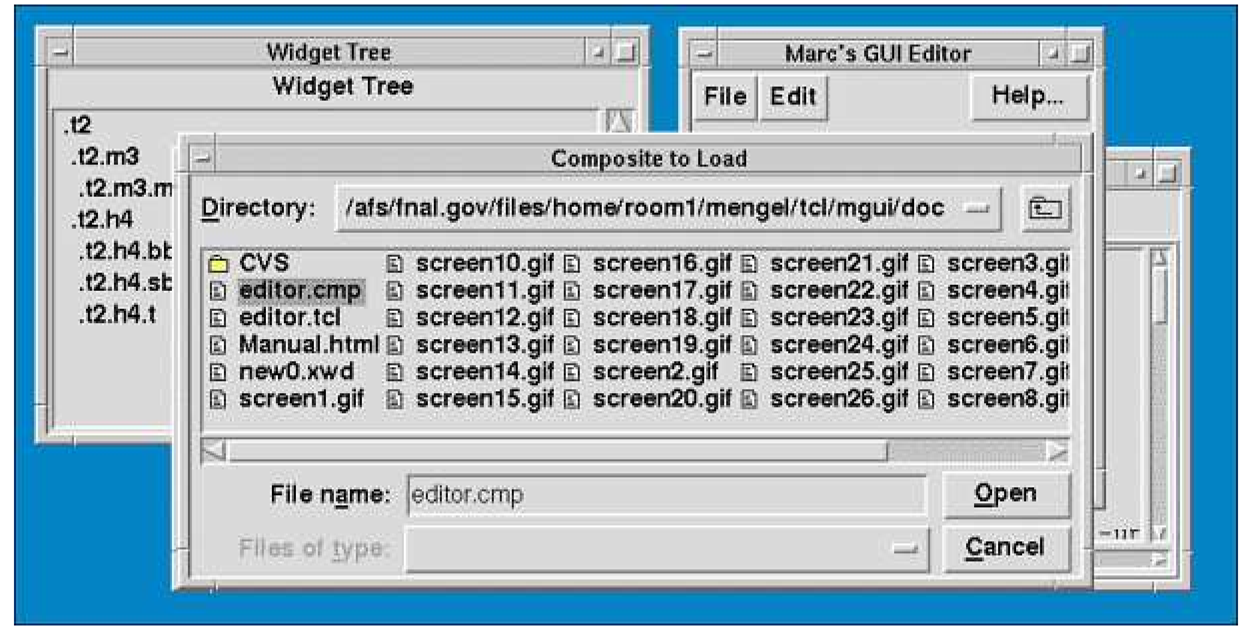


2.3.7 Loading and using the composite

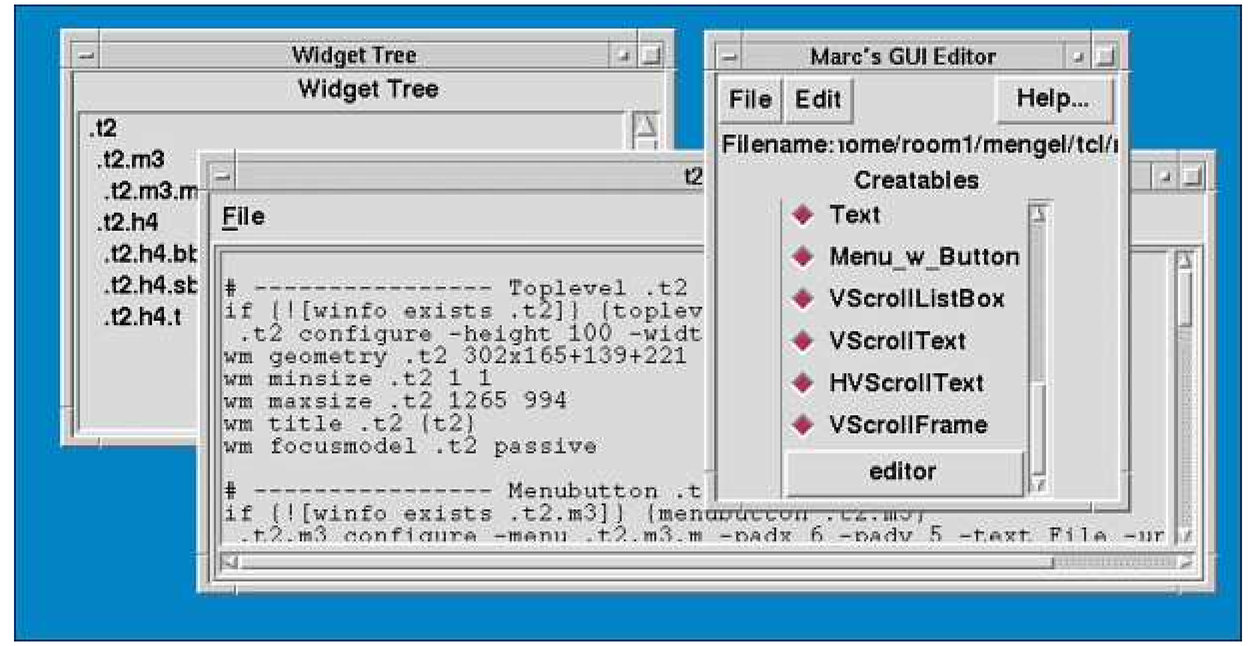
To load the composite back in, and use it to build bigger applications, by choosing the **Load Composite...** menu item from the **File** menu in the main **Marc's GUI Editor** window.



Choose the "editor.cmp" file from the file browser, and hit **Open**



Now if you scroll down the **Creatables** window, the new **editor** button appears,
and you can simply press it to create more editors.



Prev (cf. Section 2.3.5)Making Resizing Work

3 Importing interfaces into mgui

You can import the interface from an exisiting Tcl/Tk script simply by loading it into mgui with the "Load" button. However, when you save it, only the *interface* code will be saved.

Any procedures or variables defined in the imported script are ignored. If you Save under the same filename, you will lose any such code.

4 Advanced mgui

There are several features of mgui that are provided for advanced and heavy duty users, or are experimental.

- "gang" widget configuration
- composite widget creation
- export to non-Tcl Tk environments (perlTk, python Tkinter)

4.1 Gang widget configuration

4.1.1 putting widgets in gangs

You can tag widgets into one of 26 groups by positioning the pointer over the widget and using the key sequence <Key-Esc><Key-m><Key-letter> where letter is the name of the group. This will create a small green button labelled *letter* on the widget.

Currently a button may be in only one gang at a time.

Note that the keypress activiation occurs separately from the Tk input focus, but *not* separately from the X keyboard focus; that is the X keyboard focus must be on some window in the Tk application running mgui, but the widget in question does not have to be highlighted with the Tk input focus to be tagged.

4.1.2 configuring gangs

Pressing the green letter-ed button created when you tagged the widget into a gang gives you a configure widget for that gang. Items you change and press <Key-Return> on, or that you change from the menu will be applied to the whole "gang". Pressing the "Apply" button will cause *all* of the options in the configuration popup to be applied to *all* of the widgets in the gang, making them largely identical.

4.2 Composite widget creation

Using the popup menu gives you the choice of saving a subtree of the widget set as a "composite". This composite can then be loaded back in to make a template, much like the "HVScrolledText" and other composites that come with **mgui**. The Introduction to mgui (cf. Section 2) shows how to use this to make cookie-cutter style templates you can use to make large numbers of similar widget clusters.

However, the subroutine file that this generates is a good way to generate pop-ups for applications you are building, or to make simple mega-widgets that you can create dynamically as a result of changing conditions, directory contents, etc.

4.3 Non-Tcl versions of Tk

mgui has early support for generating perlTk and python scripts. While it cannot *Load* such scripts, it can write them, or at least a reasonable facsimile thereof. Currently the scripts generated for those languages often requre some cleanup by hand to get the desired behavior.