0pt=0.4pt 0.4pt=0.4pt

BEAM GUI Maintenance Guide

J. Treurniet
Ionizing Radiation Standards
National Research Council of Canada
Ottawa, K1A OR6
Printed: August 30, 2019
NRCC Report PIRS-0509(A)revB

The main window of the BEAM GUI.

Abstract

This document is intended for anyone who will be making changes to the BEAM or DOSXYZ GUI code. It contains variable naming conventions, a list of procedures and what they do, and a flow diagram of how they interact with each other.

Contents

Chapter 1

BEAM GUI

1.1 Introduction

Currently the working source code for the GUIs are stored in `$OMEGA_HOME/progs/gui/beam` and `$OMEGA_HOME/progs/gui/dosxyz`. They are written in Tcl/Tk. On this system, the versions available are Tk version 4.1 and Tcl version 7.5 with wish version 4.1. These are older versions. Note that Tcl/Tk is not always backward compatible, but is more backward than forward compatible. We also have wish8.0 (neotcl8.0) but the "wish" symbolic link points to wish4.1. At this time, the GUIs are not portable to Windows or Mac because when directory trees are specified I did not always use the 'file join' command but simply put in forward slashes, '/'. This doesn't matter because BEAM only runs on UNIX-based systems. It might be fun to try it out though...

1.2 Tcl/Tk Notes

Tcl is a scripting language and is similar in syntax to C-shell scripts. When referencing a variable which has been set (using `set var 5` for example), the dollar sign is used: `set newvar $var`.

Tcl does just about everything in strings. '=' is not used. If you want to set a value, you have to use `set var 6`. It manages to regard `$var` as an integer if for loops, but if you want to do any math with it you have to use the 'expr' command, for example, `set newvar [expr 5 * $var]`.

One problem I encountered was reading an integer from a file. Say that integer is 3, and I read it as 'var' then I want to compare it: `if { $var == 3 } { ...` This statement may come back as false if there are trailing spaces in the file. To fix this, use `set var [expr $var]`.

Arrays can be created in Tcl, but the important note is that the index of the array is regarded as a string, not as an integer. For example, an array can be `array(3)` or `array(colour)`. Two-dimensional arrays can only be emulated by using `array(2,1)` for

example, which would NOT be the same thing as `array(2, 1)`.

1.2.1 Previews

The canvas widget is used for all previews. A main canvas is put on the window first. If there is more than one perspective to be shown, new canvas widgets are created, then placed on the main canvas after being drawn on. Otherwise the drawing is done on the main canvas. The CM is drawn on first, in the appropriate scale. Then four rectangles are drawn on the canvas to cover up the parts of the CM drawing that stray beyond the drawing region (procedure coverup). Then the axes are drawn on (procedure draw_axes).

The preview of the entire accelerator (draw_accel) either uses only the xz cross-sectional views for each component or the yz cross-sectional views. For that reason, all xz cross-section procedures are named add_$cmname and all yz cross-section procedures are named add_$cmname_yz. Of course, if the CM is radially symmetric, like SLABS CONS3R CONESTAK FLATFILT and CHAMBER, there is no difference between yz and xz cross-sections so there's only one precedure, add_$cmname, for both. The xy cross-section procedure is add_$cmname_xy. For the MLC preview, since the leaves could be parallel or perendicular to x, the procedured are names add_MLC_sides and add_MLC_ends and are treated as a special case.

1.2.2 Translating an (x,y) point onto canvas coordinates

Let the real axes have ranges (x_{min},x_{max}) and (y_{min},y_{max}) and let the canvas be $n \times n$. Then for the x coordinate,

$$x_c = mx_r + b$$

Using the translation of the origin of the real coordiante system, $(0,0)_c \rightarrow (x_{min}, y_{min})_r$,

$$x_c = mx_{min} + b = 0$$

so that $b = -mx_{min}$. The using the translation of the maximum, $(n,n)_c \rightarrow (x_{max}, y_{max})_r$,

$$x_c = mx_{max} + b = mx_{max} - mx_{min} = n$$

so that $m = n/(x_{max} - x_{min})$. Putting it all together,

$$x_c = n\frac{(x_r - x_{min})}{(x_{max} - x_{min})}$$

Note that the origin on the canvas is in the upper left-hand corner. To shift the entire graph on the canvas to an origin of (l,m) simply add these to x and y:

$$x_c = n\frac{(x_r - x_{min})}{(x_{max} - x_{min})} + l$$

and similarly for the y-coordinate,

$$y_c = n\frac{(y_r - y_{min})}{(y_{max} - y_{min})} + m$$

1.3 The BEAM GUI

1.4 Tcl/Tk Files for the BEAM GUI

The files required for the GUI are the following:

1. `beam_gui` is the main script. It pops up the initial window. The main window presents the user with four options,

   - load a previous accelerator – allows the user to browse their .module files to select an accelerator to work with.
   - specify a new accelerator – allows the user to create a new accelerator by specifying the CMs they intend to use as well as giving them identifying names.
   - load a previous input file – allows the user to browse their home directory for a previously made input file. If an accelerator has been loaded, an attempt is made to start the browser off in the directory corresponding to the accelerator selected. If that is not possible, or if an accelerator has not yet been loaded, the browser starts in the `$home/egs4` directory.

     If an accelerator has not been loaded and an input file is selected, an attempt is made to load the appropriate accelerator based on the directory tree.
   - save input parameters as... – allows the user to browse directories and specify an input file name to save the input parameters as.

   (a) proc get_beam_user_defaults {} reads in default maximum and minimum parameter values from `beam_user_macros.mortran`.

   (b) about_BEAM_GUI {} pops up a window describing the origin of the GUI.

   (c) about_BEAM {} pops up a window describing the origin of BEAM.

   (d) exit_prompt {} puts an "Are you sure" dialog up before exiting.

2. `beam_params.tcl` contains parameter settings for the GUI, like parameter names and initial values and the various options for those parameters with multiple options. There are no procedures in this file.

3. `create_accel.tcl` contains procedures which allow the user to build a new accelerator and procedures to display it or previously built accelerator on the screen.

   (a) `specify_new_module` {} calls create_accel after checking whether an accelerator is already loaded and prompting to discard it if necessary. It is called from the main File menu, "specify a new accelerator".

   (b) popup_cm_window {} creates a toplevel window to hold the labels and buttons which display the selected CMs.

   (c) update_popup {} is called when a new CM is added or when a module file is finished loading. It puts the CM name, identifier and Edit button on the popup window.

(d) edit_cm { id index } is called when the "Edit" button is pressed for a CM. id is the chronological CM number and index is cm_type($id) The Z-coordinate of the end of the previous CM is calculated by calling get_zmax (in `draw_accel.tcl`). That number is then passed to the CM being edited so that the user knows where the last CM ended.

(e) create_accel {} is the procedure used to pop up a window so that the user can select form the available CMs, give it an identifier name, and add it to the popup window using add_cm.

(f) seeifsaved {} is used when the user was working on a new accelerator and tries to exit without saving. It prompts with an "Are you sure?"

(g) put_cm_in_textbox {} is used with a double-click binding with create_accel, to automatically insert a CM name into the selected textbox beside the list.

(h) add_cm {} is used to add a CM to the list selected for the accelerator. It is called by create_accel and in turn calls update_popup. It sets the variables for selected CMs.

4. `query_filename.tcl` contains the procedure for browsing a directory tree for a file.

   (a) query_filename {next_proc_name dir ext} is the main procedure. `next_proc_name` is the procedure to follow when the "OK" button is pressed, `dir` is the starting directory for the search, and `ext` is a file extension, which serves as a filter.

   (b) call_next_proc { next_proc_name } is the procedure called when the "OK" button is clicked. It calls `next_proc_name`.

5. `browser.tcl` is similar to query_filename but does not use a filter and always starts in the user's egs4 directory. `next_proc_name` is always one of the four 'set_*' listed below.

   (a) browse {next_proc_name} pops up a browser with no filter.

   (b) set_spcnam15 {} sets `spcnam15` to the file selected in browse.

   (c) set_spcnam21 {} sets `spcnam21` to the file selected in browse.

   (d) set_spcnam31 {} sets `spcnam31` to the file selected in browse.

   (e) set_spec_file {} sets `spec_file` to the file selected in browse.

6. `save_module.tcl` contains procedures for saving a module file (some of which are also used for saving an input file).

   (a) save_module {} is the procedure for saving an accelerator. It calls query_filename to obtain a module filename, with arguments `DoesModFileExist egs4/beam/spec_modules module`.

   (b) DoesModFileExist { tree filename } checks whether the module file selected already exists, and if so, it calls overwrite_prompt to prompt the user before overwriting. save_proc is save_mod_file and type is 'mod', used in overwrite_prompt.

   (c) save_mod_file {tree filen} saves the module file to the file `tree/filen`.

   (d) overwrite_prompt {tree filename save_proc type} prompts the user as to whether it's okay to overwrite the file `tree/filename`, if it already exists. type is either 'mod' or 'inp', for module or input.

7. `load_input2.tcl` contains procedures for loading a previously made input file.

   (a) load_input {} attempts to locate the appropriate starting directory based on the module file loaded, if any, and then calls query_filename with set_inp_file as the `next_proc_name`.

   (b) set_inp_file {tree filen} sets the input file name `inp_file` to that selected in query_filename, `tree/filen`. If there is no accelerator loaded, however, it tries to locate the appropriate module file. If it can't find it, the user is alerted that an accelerator must be loaded first.

   (c) get_val { data varname i } scans the string data for the first number in the line and sets varname($i) to this value.

   (d) get_str { data varname i } gets a string from data and trims the whitespace off either end, and sets varname($i) to this value.

   (e) read_input {} reads in the input file. It uses the information loaded from the module file to read the CM parameters.

8. `load_module.tcl` contains procedures to load a previously defined accelerator.

   (a) load_old_module {} checks whether an accelerator is already loaded, and if so, prompts the user as to whether it should be discarded. It calls query_filename with set_mod_file as the next procedure.

   (b) set_mod_file {tree filen} sets `mod_file` to `tree/filen`, then calls load_module to read in the information.

   (c) load_module {} reads in the accelerator information, *i.e.* the CM names and identifiers (`cm_name(i), cm_ident(i)`).

9. `pegs4.tcl` contains procedures for loading a PEGS4 file.

   (a) get_pegs4file {} pops up a window to select the PEGS4 file. The user has to select it by browsing (which calls query_filename).

   (b) set_pegs4filename { tree filename } sets the `pegs4file` to `tree/filename` and configures the label that displays the directory it was selected from.

   (c) check_pegs4file {} If the file was selected from the `HEN_HOUSE`, the user's home directory is searched for a matching file. If found, a dialog box appears informing the user that they'll have to reselect the file in their home directory, since BEAM will search this first.

10. `set_main_inputs.tcl`

    (a) set_main_inputs {} first checks whether an accelerator has been loaded and if not it pops up an info window to tell the user to load or create one first before the main inputs can be set. Otherwise, it pops up the main window for setting main inputs, which puts the parameter labels (`names($i)`) and text boxes/option menus up on the window. Any parameters which are off-shoots of these are dealt with on a child window.

(b) set_medium { iopt } sets the medium (`values(2)`) to `medium(iopt)` and configures the option menu to show this value.

(c) make_menu_button {i w} creates an option menu for parameter i on frame w (left or right side of the main window). The command `set_value` is associated with each option.

(d) make_text_box {i w} creates a text entry box for parameter i on frame w (left or right side of the main window).

(e) set_value {io iopt w} is used with the option menu command. `io` is the parameter index (*i.e.* `names($io)`), `iopt` is the index of the option selected and w is the menu widget it came from. Generally, this prcedure sets the value of the parameter to that selected by the user and configures the option menu widget to display this choice. For those parameters which require setting further parameter values, another procedure is called from here. These are listed below.

(f) set_nbrem {} asks the user for the number of brem photons, when uniform brem splitting is selected, and also FS, SSD, NMIN when selective brem splitting is selected. In both cases, gives the Russian Roulette option menu.

(g) set_special_N {} is used with the special case of IWATCH; it asks for the history at which to set IWATCH to 2.

(h) set_scoring_options { nplanes } is an offshoot of the number of scoring planes (nplanes). It gets `IPLANE_TO_CM` and the number of scoring zones.

(i) get_zone_marks { w izone } is called from set_scoring_options to get the zone marks after the number of zones in input.

(j) set_28_3 { i w label } sets the value of `values(28,3,$i)` to `$label` and configures `$w.inp` to display it. This value is the zone type and is called from set_scoring_options.

(k) set_itdose_on {} is called when dose calculations are selected. It asks for the contaminant type, the CM at which to identify it as contaminant, and the number of inclusive and exclusive bit filters desired.

(l) configure_menu { w i j k } configures widget `$w.inp` to display `$options($i,$j,$k)` on it. It is exclusively for use in set_itdose_on to set the contaminant type.

(m) set_ln_exc {} is called when the number of dose components with exclusive bit filters are defined. It uses radiobuttons to select bits.

(n) set_ln_inc {} is called when the number of dose components with inclusive bit filters are defined. It uses radiobuttons to select bits.

(o) set_force_options {} is called when photon forcing is set to on. A window pops up asking for the four options associated with this parameter.

(p) set_ifluor_options { level } is called when x-ray fluorescence is set to on. It asks for the effective Z, from and to regions.

(q) set_src_options { iopt } is called when source `iopt` is selected. Note that iopt does not follow the numbering assigned to the sources in the user manual but are chronological in the same order as in the manual. The user is asked to set the source options associated with the source selected (number of options and option

names are set in `beam_params.tcl`). Any further options, such as spectrum files or beam energies, are also there.

(r) get_s9vals { w } pops up a window with text boxes so that the user can input the discrete probabilities for source 9.

(s) set_ioutsp {label} is called when the variable ioutsp is set on the main source options window. It configures the option menu to display the new value and sets the parameter value.

11. `save_input.tcl` contains procedures to save an input file.

    (a) save_input {} is the main procedure for saving an input file. It calls query_filename to obtain an input filename, with arguments `DoesModFileExist $inp_file_dir egs4inp`. If `$inp_file_dir` doesn't exit, it asks the user if they want to create it.

    (b) DoesInpFileExist {} checks to see whether the file exists. If it does, it calls overwrite_prompt. It then calls save_inp_file.

    (c) save_inp_file {tree filen} sets the variable `inp_file` to `tree/filen`, then calls create_file to generate the file.

12. `help.tcl` contains all of the help windows. The first procedure is help, which calls help_dialog to pop up a window to display the help text for the main input parameters. The second procedure is help_srcopts, which calls help_gif to pop up a window to display the gif file and help text for the sources. The rest of the procedures have names that start with help_ and the rest of the name is (hopefully) obvious.

13. `new_create_file.tcl` contains the procedures used to write an input file.

    (a) create_file { file } writes the main parameters and the parameters defined for the selected CMs to a file in the specified format. It is called from save_inp_file.

14. `misc.tcl` contains dialog boxes and macro-ish procedures that are used alot in the CM procedures (ecut, pcut, etc.).

    (a) tk_dialog { w title text bitmap default args } overrides the tk pre-made procedure. I increased the width of the box and changed the font to helvetica.

    (b) help_dialog { w title text bitmap default args } creates a window with a scrollbar to display `text`. The scrollbar is required for those help texts with more than 30 lines.

    (c) help_gif { w text iconfile } creates a window with `iconfile` (a gif file) displayed on the left and `text` on the right. It is used for the source helps and the CM helps.

    (d) get_1_default { name top text } is used when there is one default value defined in `$name_macros.mortran` and displays it at the top of the main edit window for all CMs.

    (e) get_2_defaults { name top str1 text1 str2 text2 } is used when there are two default values defined in `$name_macros.mortran` and displays them at the top of the main edit window for all CMs.

(f) add_rmax_square { w index } adds the block of helpbutton, label, textbox for RMAX_CM for a square CM.

(g) add_rmax_rad { w index } adds the block of helpbutton, label, textbox for RMAX_CM for a radially symmetric CM.

(h) add_title { w index } adds the block of helpbutton, label, textbox for TITLE.

(i) add_zmin { w index } adds the block of helpbutton, label, textbox for ZMIN.

(j) add_ecut { w index } adds the block of helpbutton, label, textbox for ECUT.

(k) add_pcut { w index } adds the block of helpbutton, label, textbox for PCUT.

(l) add_dose { w index } adds the block of helpbutton, label, textbox for DOSE_ZONE.

(m) add_latch { w index } adds the block of helpbutton, label, textbox for LATCH.

(n) add_material { w index } adds the block of helpbutton, label, textbox for MATERIAL.

(o) set_material { w iopt index } sets the material for the add_material option menu.

15. change_color_scheme.tcl contains procedures for the user to change the colour scheme for drawing. A default colour scheme is set up in beam_gui, but greyscale can be selected here, and by clicking on a colour swatch the user can alter the rgb values of a colour using scale widgets.

(a) change_color_scheme {} pops up a window with the colours on 10x10 canvases on a grid with the materials they correspond to.

(b) change_to_colour {} resets colorlist when one of the radiobuttons, colour or greyscale, is selected.

(c) change_color { parent_w i } pops up a window with the rgb scale widgets (or with one scale widget if greyscale selected) and a canvas to display the colour.

(d) update_palette { parent_w col value } changes the colour of the canvas on the window with the rgb scales when the scales are moved.

(e) set_color { parent_w i } is called when the Accept button is selected on the window with the rgb scales; it sets the new value for the colour in colorlist (uses lrm).

(f) lrm { list pos item } replaces element at pos in list with item.

16. draw_accel.tcl contains procedures for the preview of the entire accelerator, xz cross-section.

(a) draw_accel {} pops up a window with a colour legend and buttons on the left and a canvas in scrollbars on the right. The colour swatches on the legend can be clicked to edit the colour scheme; only the materials used are shown.

(b) redraw_accel {} draws the accelerator to scale on the canvas.

(c) change_accel_range {} pops up a window with x and z ticks, zoom scales and ranges so that the user can customize the plot.

(d) get_zmax { id } finds the maximum z of the CM at position id in the accelerator.

17. slabs.tcl

(a) init_SLABS { id } initializes the parameters for SLABS, the CM at index $id.

(b) read_SLABS { fileid id } reads in the SLABS section of an input file $fileid, to be stored as the CM at index $id.

(c) edit_SLABS { id } pops up the window for editing SLABS parameters for the CM at index $id.

(d) define_nslabs { id } is an offshoot window to define the layers.

(e) write_SLABS { fileid id } writes the parameter set for the SLABS CM at index $id to file $fileid. It is called from create_file.

(f) show_SLABS { id } pops up a window in which to display the canvas for the preview of the CM, and assigns colours to the media used.

(g) draw_SLABS { id } creates the canvas (or canvasses, depending on the CM) required for the preview.

(h) add_SLABS { id xscale zscale xmin zmin l m parent_w } draws the graphic on parent_w. l is the space on the left of the graph and m is the space from the top of the graph. xmin and zmin are the minimum axis ranges, xscale and zscale are the scale factors for x and z.

(i) coverup { l m width canvas } puts four white rectangles on canvas to cover up the drawing parts that go beyond the drawing area. width is the width and height of the drawing area, l and m are as above.

(j) add_axes { pair n1 n2 l m width a b c d scale1 scale2 canvas } draws the axes on canvas. pair can be xy, xz or yz. n1 and n2 are the number of ticks to put on the 1st and 2nd axis respectively. width, l and m are as above. a, b, c and d are the pairs of min and max values for the 1st and 2nd axes, respectively.

(k) change_cm_range { id parent xyz } pops up a window to change the axis ranges and the number of ticks on the axes. Used by ALL CM previews. parent is the canvas that's being changed and is used when Apply is selected to redraw based on these changes. xyz can be xyz, xy, xz, or yz, depending on how many perspectives are displayed.

Unless explicitly stated, the init, read, edit, write, show, draw and add procedures have a description similar to that stated in SLABS.

18. `cons3r.tcl`

(a) init_CONS3R { id }

(b) read_CONS3R { fileid id }

(c) edit_CONS3R { id }

(d) define_nnode { id } pops up a window for the definition of the points used in this CM.

(e) write_CONS3R { fileid id }

(f) show_CONS3R { id }

(g) draw_CONS3R { id }

(h) add_CONS3R { id xscale zscale xmin zmin l m parent_w }

19. `conestak.tcl`

(a) init_CONESTAK { id }

(b) read_CONESTAK { fileid id }

(c) edit_CONESTAK { id }

(d) disable_if_off { id } is used to disable the input area for the outer wall if it has been selected as not present. It is called when the checkbutton is clicked and uses the variable owall_conestak($id).

(e) define_conestak_props { id } pops up a window for defining the properties of the layers, inside and outside.

(f) define_conestak_dim { id } pops up a window for defining the dimensions of each layer (thickness, top radius, back radius).

(g) write_CONESTAK { fileid id }

(h) show_CONESTAK { id }

(i) draw_CONESTAK { id }

(j) add_CONESTAK { id xscale zscale xmin zmin l m parent_w }

20. flatfilt.tcl

(a) init_FLATFILT { id }

(b) read_FLATFILT { fileid id }

(c) edit_FLATFILT { id }

(d) define_layers { id }: if there is more than one window required (more than 9 layers), a child pops up to hold buttons for "Edit layers 1-9, 10-18, etc.", which connects to procedure define_layer_window for those layers. Otherwise we go directly to define_layer_window for all layers.

(e) set_flatfilt_button_color { id ilayer start end } determines whether all of the necessary parameters have been set for layers `start` to `end`, corresponding to button `ilayer`. If they haven't, the button colour is set to red, otherwise black.

(f) define_layer_window { id } pops up a window with thickness and number of cones in the layer for each layer. A "Define conical sections >>" button leads to define_cones.

(g) define_cones { parent id indx } pops up a window for layer `indx` so that the user can set the geometry and properties of each cone in the layer.

(h) write_FLATFILT { fileid id }

(i) show_FLATFILT { id }

(j) draw_FLATFILT { id }

(k) add_FLATFILT { id xscale zscale xmin zmin l m parent_w }

21. jaws.tcl

(a) init_JAWS { id }

(b) read_JAWS { fileid id }

(c) edit_JAWS { id }

(d) define_jaws { id } pops up a window for defining the geometric and material properties of each set of paired bars.

(e) write_JAWS { fileid id }

(f) show_JAWS { id }

(g) draw_JAWS { id }

(h) add_JAWS { id xscale zscale xmin zmin l m parent_w }

(i) add_JAWS_yz { id yscale zscale ymin zmin l m parent_w }

22. circapp.tcl

(a) init_CIRCAPP { id }

(b) read_CIRCAPP { fileid id }

(c) edit_CIRCAPP { id }

(d) define_circapp { id } pops up a window for defining geometry and material properties of each scraper.

(e) write_CIRCAPP { fileid id }

(f) show_CIRCAPP { id }

(g) draw_CIRCAPP { id }

(h) add_CIRCAPP { id xscale zscale xmin zmin l m parent_w }

(i) add_CIRCAPP_xy { id xscale yscale xmin ymin l m parent_w }

(j) add_CIRCAPP_yz { id yscale zscale ymin zmin l m parent_w }

23. applicat.tcl

(a) init_APPLICAT { id }

(b) read_APPLICAT { fileid id }

(c) edit_APPLICAT { id }

(d) define_applicat { id } pops up a window for defining geometry and material properties of each scraper.

(e) write_APPLICAT { fileid id }

(f) show_APPLICAT { id }

(g) draw_APPLICAT { id }

(h) add_APPLICAT { id xscale zscale xmin zmin l m parent_w }

(i) add_APPLICAT_xy { id xscale yscale xmin ymin l m parent_w }

(j) add_APPLICAT_yz { id yscale zscale ymin zmin l m parent_w }

24. pyramids.tcl

(a) init_PYRAMIDS { id }

(b) read_PYRAMIDS { fileid id }

(c) edit_PYRAMIDS { id }

(d) define_pyramids { id } pops up a window for defining geometry and material properties of each layer.

(e) set_grid_material { w iopt index } configures the option menu `w` in the grid on the window created in define_pyramids to a value of `cmval(index)` and sets `cmval(index)` to `medium(iopt)`.

(f) write_PYRAMIDS { fileid id }

(g) show_PYRAMIDS { id }

(h) draw_PYRAMIDS { id }

(i) add_PYRAMIDS { id xscale zscale xmin zmin l m parent_w }

(j) add_PYRAMIDS_yz { id yscale zscale ymin zmin l m parent_w }

25. chamber.tcl

(a) init_CHAMBER { id }

(b) read_CHAMBER { fileid id }

(c) edit_CHAMBER { id }

(d) define_chamber_top_props { id } pops up a window for defining the material and properites of each layer in the top of the chamber.

(e) define_chamber_bot_props { id } pops up a window for defining the material and properites of each layer in the bottom of the chamber.

(f) define_chamber_props { id } pops up a window for defining the material and properites of each layer in the central region of the chamber.

(g) set_button_color { id i } sets the "edit layer" button to red if the thickness or material have not been defined for layer i.

(h) edit_group_thick { id } is used to set the thicknesses of groups when defining the chamber as groups.

(i) edit_layer { id layer }

(j) calculate_distance { id layer } is used to calculate the distance from the top of chamber to the current layer.

(k) set_same_as_layer { id layer } is used to set the current layer to have the same values as $sameaslayer (a global variable).

(l) write_CHAMBER { fileid id }

(m) show_CHAMBER { id }

(n) draw_CHAMBER { id }

(o) add_CHAMBER { id xscale zscale xmin zmin l m parent_w }

26. mirror.tcl

(a) init_MIRROR { id }

(b) read_MIRROR { fileid id }

(c) edit_MIRROR { id }

(d) calc_mirror_angle { id } is used to calculate the angle of the mirror given the geometry specified.

(e) define_mirrors { id } pops up a window to define the thickness and material properties of each layer in the mirror.

(f) write_MIRROR { fileid id }

(g) show_MIRROR { id }

(h) draw_MIRROR { id }

(i) add_MIRROR { id xscale zscale xmin zmin l m parent_w }

27. mesh.tcl

(a) init_MESH { id }

(b) read_MESH { fileid id }

(c) edit_MESH { id }

(d) write_MESH { fileid id }

(e) show_MESH { id }

(f) draw_MESH { id }

(g) add_MESH { id xscale zscale xmin zmin l m parent_w }

(h) add_MESH_xy { id xscale yscale xmin ymin l m parent_w }

28. xtube.tcl

(a) init_XTUBE { id }

(b) read_XTUBE { fileid id }

(c) edit_XTUBE { id }

(d) define_xtube { id } pops up a window to define the thickness and material properties of each layer in the x-ray tube.

(e) write_XTUBE { fileid id }

(f) show_XTUBE { id }

(g) draw_XTUBE { id }

(h) add_XTUBE { id xscale zscale xmin zmin l m parent_w }

29. sidetube.tcl

(a) init_SIDETUBE { id }

(b) read_SIDETUBE { fileid id }

(c) edit_SIDETUBE { id }

(d) define_sidetube { id } pops up a window to define the outer radius and material properties of each layer in the sidetube.

(e) write_SIDETUBE { fileid id }

(f) show_SIDETUBE { id }

(g) draw_SIDETUBE { id }

(h) add_SIDETUBE { id xscale zscale xmin zmin l m parent_w }

(i) add_SIDETUBE_xy { id xscale yscale xmin ymin l m parent_w }

30. mlc.tcl

(a) init_MLC { id }

(b) read_MLC { fileid id }

(c) edit_MLC { id }

(d) define_leaves { id } pops up a window to define the size of the leaves in the mlc, in groups.

(e) save_mlc { id } is used when leaving define_leaves. It checks that an error hasn't been made when defining the leaves, and if so it informs the user.

(f) add_mlc_row { id } is invoked when "Add a row" is clicked on the window in define_leaves. It adds a new group to the bottom of the list. If all the leaves are currently defined, it does not allow a new group.

(g) write_MLC { fileid id }

(h) show_MLC { id }

(i) draw_MLC { id }

(j) add_MLC_ends { id xscale zscale xmin zmin zmax l m parent_w }

(k) add_MLC_sides { id xscale zscale xmin zmin zmax l m parent_w }

(l) add_MLC_xy { id xscale yscale xmin ymax l m parent_w }

31. block.tcl

(a) init_BLOCK { id }

(b) read_BLOCK { fileid id }

(c) edit_BLOCK { id }

(d) define_block { id } is called from edit_BLOCK; it pops up a window to set the number of points which define each subregion.

(e) define_points { j id } is called from define_block; it allows the user to set the (x,y) points for each subregion.

(f) check_block_angles { id subreg }

(g) cross { x1 y1 x2 y2 x3 y3 }

(h) write_BLOCK { fileid id }

(i) show_BLOCK { id }

(j) draw_BLOCK { id }

(k) add_BLOCK { id xscale zscale xmin zmin l m parent_w }

(l) add_BLOCK_xy { id xscale yscale xmin ymax l m parent_w }

(m) print_canvas { w } pops up a window so the user can choose colour or black and white, portrait or landscape, and print to a file or to a printer.

(n) print_cmd { w } executes the print command based on what the user chose.

32. arcchm.tcl

(a) init_ARCCHM { id }

(b) read_ARCCHM { fileid id }

(c) edit_ARCCHM { id }

(d) define_arcchm_props { id } is called from edit_ARCCHM; it pops up a window to define the properties of each ion chamber and septum.

(e) write_ARCCHM { fileid id }

(f) show_ARCCHM { id }

(g) draw_ARCCHM { id }

(h) add_ARCCHM { id xscale zscale xmin zmin l m parent_w }

(i) add_ARCCHM_yz { id yscale zscale ymin zmin l m parent_w }

The last 15 files are specifically for the CMs they are named for. The init_$name, read_$name, edit_$name, write_$name, show_$name and draw_$name procedures are basically the same for each CM so a description is not provided for every one, only SLABS.

The procedures described above are shown as a flow diagram in figure 2.1, excluding those procedures used in editing input parameters. When the "done" state is reached, the `.cm_selected` toplevel window has been popped up, with an "Edit main parameters" button, which calls the procedure set_main_parameters, and `id` (one for each CM selected) frames with labels `$cm_names($cm_type($id))` (to show the CM type) and `$cm_ident($id)` (to show the CM identifier), and an "Edit" button, which calls the procedure edit_cm, which calculates the Z-coordinate of the end of the previous CM and then calls edit_$cm_names($cm_type($id)) with argument `id`.

1.5 Adding a Component Module to the Package

Let the new CM be called "NEWCM". Since there are 16 now we'll say it's the 17^{th} CM.

1. In `beam_params.tcl`, add "NEWCM" to the end of the array `cm_names` with `set cm_names(17) ''NEWCM''`.

2. Since there are now 17 CMs, in `create_accel.tcl` you will have to change 16 to 17 in the create_accel procedure:

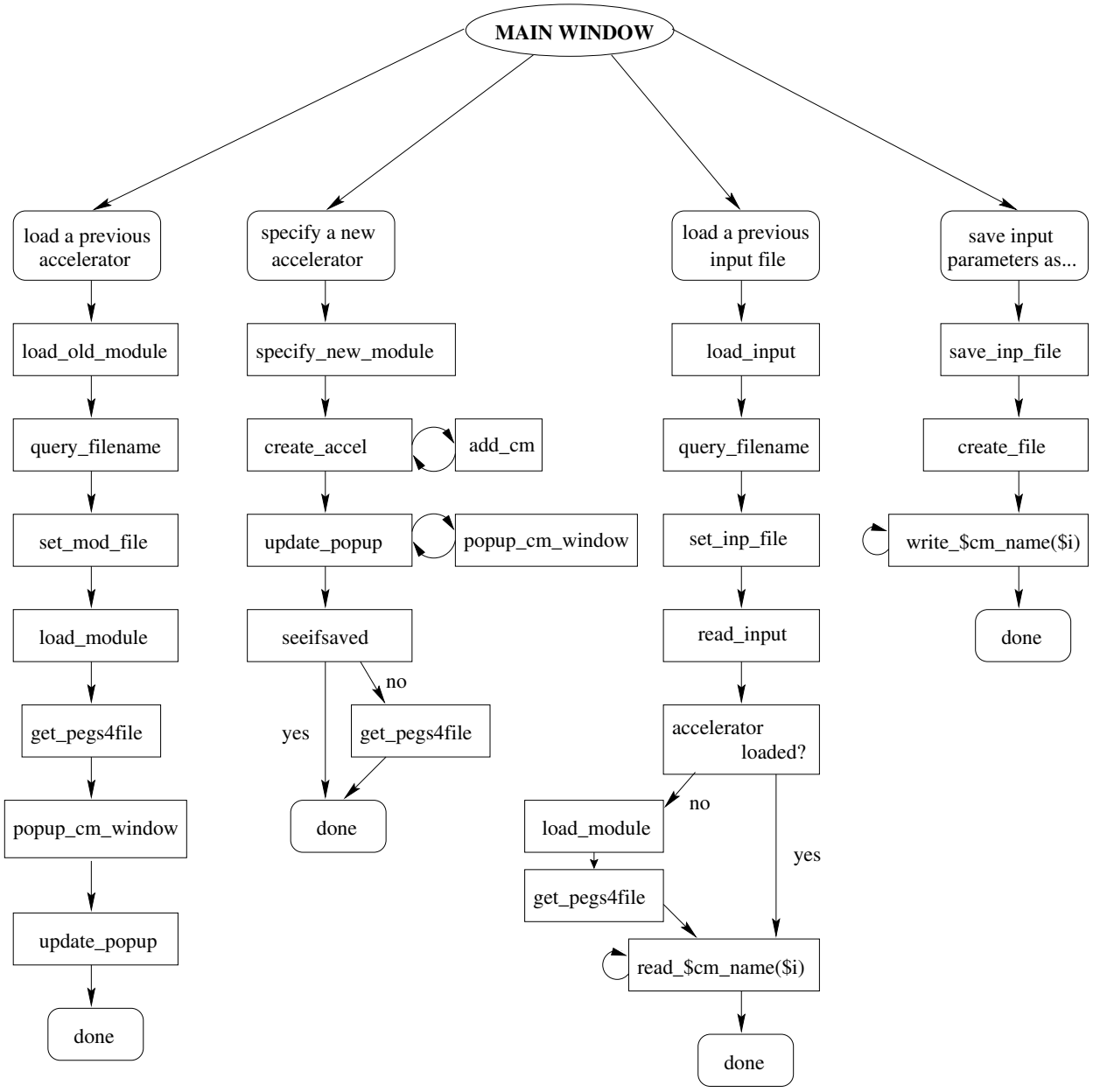

Figure 1.1: Flow diagram of the procedure accessible from the top window of the GUI. Procedure names are in rectangular boxes. Not all procedures are shown; "done" represents the point at which procedures involved in setting parameters are used.

```
# add listbox, scrollbar, insert all availabel CM names
listbox $top.l.list -height 10 -yscrollcommand "$top.l.scrl set" \
   -bg white
for {set i 1} {$i <= 16} {incr i} {
   $top.l.list insert end $cm_names($i)
}
scrollbar $top.l.scrl -command "$top.l.list yview"
pack $top.l.scrl -side right -fill y
pack $top.l.list -side left -fill x -fill y
```

and in the add_cm procedure:

```
for {set i 1} {$i<=16} {incr i} {
   if [string compare $new_cm_name $cm_names($i)]==0 {
      set index [incr i -1]
      break
   }
}
```

3. In load_module.tcl, change 16 to 17 in

```
for {set j 1} {$j <= 16} {incr j} {
    # look for a match in cm_names to get cm_type
    if [string compare $cm_names($j) $str]==0 {
        set found 1
        incr num_cm
        incr i
        set cm_type($i) $j
    }
}
```

4. In `draw_accel.tcl`, you will have to make an addition to the get_zmax procedure in order to determine the where the CM ends. What that value will be depends entirely upon the input parameters for it.

5. The biggest part of adding a new CM is making the code for it, *i.e.* newcm.tcl. To accomplish this, it is easiest to copy the code for the CM which most closely resembles it in input format to the new file and begin editing from that starting point.

1.6 Naming Conventions

The variable naming conventions are given in this section. They are tabulated with the variable name used in the BEAM code. The main parameters are those defined in the procedure set_main_parameters. For those parameters which have multiple options instead of numerical values, an option menu is created. The options available to a parameter I are defined in `beam_params.tcl` as `options(I,J)`, where J is an integer.

If a variable is presented in the tables as having forward slashes in its index, there are multiple variables, *i.e.* `s9vals(1/2/3,i)` means there are `s9vals(1,i)`, `s9vals(2,i)`, and `s9vals(3,i)` variables.

For the CMs, the variable naming follows quite closely to the way they are presented in the user manual. `cm_type` is an array which holds an integer corresponding to one of the available CMs for each of the CMs used. `cm_ident` is an array which holds identification strings for each of the CMs used. `cm_names` is an array containing strings "SLABS", "APPLICAT", *etc.*, in the order they are presented in the manual, for use with `cm_type`. Then for the $idth CM in the accelerator, `cm_type($id)` is a number corresponding to what type of CM $id is in `cm_names` and `cm_ident($id)` is its identifying string. So if CM #2 is a SLABS CM, to get the string "SLABS", you would use `$cm_names($cm_type(2))`.

1.6.1 GUI variables

rootx	the x-coordinate of the upper left corner of the first window
rooty	the y-coordinate of the upper left corner of the first window
helvfont	helvetica font
start_dir	the directory the GUI was started in
inp_file	the full input filename (including directory tree)
inp_base	the base name of the input file (no tree, no extension)
inp_file_dir	the directory of the input file
mod_file	the full name of the module file
mod_base	the base name of the module file
pegs4filename	the full name of the PEGS4 data file (including tree)
colorlist	a list of colors to be associated with meida in drawing
medium(i), i=1,nmed	media available from PEGS4 file
nmed	the number of media read in from the PEGS4 file
num_cm	the number of CMs used
cm_names	an array of CM names
cm_ident	an array to hold the identifying strings for each CM
cm_type	an array to hold numbers which correspond to what type of CM it is

1.6.2 Main parameters

values(1)	TITLE
values(2)	MEDIUM for nominal air
values(3)	IWATCH
values(4)	ISTORE
values(5)	IRESTART
values(6)	IO_OPT
values(7)	IDAT
values(8)	LATCH_OPTION
values(9)	IZLAST
values(10)	NCASE
values(11)	IXX
values(12)	JXX
values(13)	TIMMAX
values(14)	IBRSPL, Brem splitting (none, uniform, selective)
values(15)	NBRSPL
sbrem(1/2/3)	FS, SSD, NMIN
values(16)	IRRLTT
values(17)	IQIN, Incident particle
values(18)	ISOURC, Source number
srcopts(1/2/3/4)	Four source options maximum for any source
s1opt	source 1 option, circular or square
s3opt	source 3 option, vertical or horizontal
s9vals(1/2/3,i)	Source 9 options
spcnam15	Filename for source 15
spcnam21	Filename for source 21
spcnam31	Filename for source 31
monoen	Monoenergetic or spectrum source
Ein_val	Kinetic energy of source, EIN
spec_file	Spectrum filename, FILNAM
ioutsp	IOUTSP
values(19)	ESTEPE
values(20)	SMAX
values(21)	ECUTIN
values(22)	PCUTIN
vaules(23)	IDORAY
values(24)	IREJCT_GLOBAL
values(25)	ESAVE_GLOBAL
values(26)	IFLUOR
ifluor_opts(1,i)	effective Z, IZ
ifluor_opts(2,i)	from region, IREGLO
ifluor_opts(3,i)	to region, IREGHI
values(27)	IFORCE
force_bdd(1/2/3/4)	NFMIN,NFMAX,NFCMIN,NFCMAX

values(28)	NSC_PLANES
values(28,1,i)	CM number for plane iscore, IPLANE_to_CM(I)
values(28,2,iscore)	NSC_ZONES(ISCORE)
values(28,3,iscore)	MZONE_TYPE(ISCORE)
values(28,4,iscore,i)	(RSCORE_ZONE(ISCORE,I), I=1,NSC_ZONES)
values(29)	ITDOSE_ON
values(29,1/2)	ICM_CONTAM, IQ_CONTAM
values(29,3)	LNEXC
l_n_exc	L_N_EXC(I,J), J=1, 31
exc	Used to hold an on/off array for exclusive filters
values(29,4)	LNINC
l_n_inc	L_N_INC(I,J), J=1, 31
inc	Used to hold an on/off array for inclusive filters
values(30)	Z_min_CM(1)
values(40)	ICM_SPLIT

1.6.3 APPLICAT

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2)	ZBACK
cmval(id,3,0/1)	N, ISHAPE
cmval(id,4,0/1/.../7,I)	ZMIN(I), ZTHICK(I), XMIN(I), YMIN(I), WIDTHX(I), WIDTHY(I), DOSE_ZONE, IREGION_TO_BIT
cmval(id,6,0/1/2/3)	ECUT, PCUT, DOSE_ZONE_AIR, IREGION_TO_BIT_AIR
cmval(id,7,I)	MED_IN

1.6.4 BLOCK

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2,0/1/2)	ZMIN, ZMAX, ZFOCUS
cmval(id,3)	ISUB_MAX
cmval(id,4,j)	NSUB(J)
cmval(id,4,j,0/1,i)	XHI_POINT(I,J),YHI_POINT(I,J)
cmval(id,5,0/1/2/3)	XPMAX,YPMAX,XNMAX,YNMAX
cmval(id,6,0/1/2/3)	ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,7,0/1/2/3)	ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,8)	MED_IN
cmval(id,9,0/1/2/3)	ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,10)	MED_IN

1.6.5 CHAMBER

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2)	ZMIN
cmval(id,3,0/1/2)	N_TOP, N_CHM, N_BOT
top_identical	flag: 1 if top layers identical, 0 if not
chm_identical	flag: 1 if chamber layers identical, 0 if defined individually, 2 if defined in groups
bot_identical	flag: 1 if bottom layers identical, 0 if not
Top part	
cmval(id,4,0,0/1/2,i)	each layer, ZTHICK, RCYS , NFLAG
cmval(id,4,1,0/1/2/3,i)	each layer, inner cylinders, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,4,2,i)	each layer, inner cylinders, MED_IN
cmval(id,4,3,0/1/2/3,i)	each layer, outer annuli, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,4,4,i)	each layer, outer annuli, MED_IN
Chamber/phantom part	
cmval(id,5,0,0/1/2)	RCYS(1,1), RCYS(1,2), RCYS(1,3)
cmval(id,5,1,0/1,i)	each layer, ZTHICK, NFLAG
cmval(id,5,2,0/1/2/3,i)	each layer, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,5,3,i)	each layer, MED_IN
cmval(id,5,4,0/1/2/3)	chamber wall, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,5,5)	chamber wall, MED_IN
cmval(id,5,6,0/1/2/3)	gap, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,5,7)	gap, MED_IN
cmval(id,5,8,0/1/2/3)	container wall, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,5,9)	container wall, MED_IN
Bottom part	
cmval(id,6,0,0/1/2,i)	each layer, ZTHICK, RCYS , NFLAG
cmval(id,6,1,0/1/2/3,i)	each layer, inner cylinders, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,6,2,i)	each layer, inner cylinders, MED_IN
cmval(id,6,3,0/1/2/3,i)	each layer, outer annuli, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,6,4,i)	each layer, outer annuli, MED_IN
cmval(id,7)	MRNGE

1.6.6 CIRCAPP

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2)	ZBACK
cmval(id,3)	N, number of layers
cmval(id,4,0/1/.../6,i)	ZMIN(I), ZTHICK(I), ROPEN(I), XOUTER(I), YOUTER(I), DOSE_ZONE(I), IREGION_TO_BIT(I)
cmval(id,5,0/1/2/3)	ECUT, PCUT, DOSE_ZONE_AIR, IREGION_TO_BIT_AIR
cmval(id,6,i)	MED_IN

1.6.7 CONESTAK

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2,0/1)	ZMIN, RBN
owall_conestak(id)	0 for no wall, 1 for a wall
cmval(id,3)	ISCM_MAX, Number of conical layers
cmval(id,4,0/1/2,i)	each layer, ZTHICK(I), RMIN(I), RMAX(I)
cmval(id,5,0/1/2/3)	outer wall, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,6)	outer wall, MED_IN
cmval(id,7,i,0/1/2/3)	each layer inside cone, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,8,i)	each layer inside cone, MED_IN
cmval(id,9,i,0/1/2/3)	each layer outside cone, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,10,i)	each layer, outside cone, MED_IN

1.6.8 CONS3R

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2)	ZMIN
cmval(id,3)	ZTHICK
cmval(id,4)	NUM_NODE
cmval(id,5,0/1,i)	ZCORNER(I), RCORNER(I)
cmval(id,6,0,0/1/2/3/4/5)	inner ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT, IREJCTIN, MED_IN
cmval(id,6,1,0/1/2/3/4/5)	outer ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT, IREJCTIN, MED_IN

1.6.9 FLATFILT

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE (60A1)
cmval(id,2)	ZMIN
cmval(id,3)	ISCM_NO (number of layers)
cmval(id,4,0/1,i)	each layer, ISSCM_NO(I), ZTHICK(I)
cmval(id,5,i,j)	each layer, cone, RTOP(I,J)
cmval(id,6,i,j)	each layer, cone, RBOT(I,J)
cmval(id,7,0/1/2/3,i,j)	each layer, cone, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,8,j)	each cone, MED_IN

1.6.10 JAWS

cmval(id,0)	RMAX_CM(ICM_JAWS)
cmval(id,1)	TITLE
cmval(id,2)	ISCM_MAX, Number of paired bars or jaws
cmval(id,3,i)	each pair, XY_CHOICE
cmval(id,4,0/1/2/3/4/5,i)	each pair, ZMIN(I), ZMAX(I), XFP(I), XBP(I), XFN(I), XBN(I)
cmval(id,5,0/1/2/3)	interior, ECUT, PCUT, DOSE_ZONE, IREGION_to_BIT
cmval(id,6,0/1/2/3,i)	each pair, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,7,i)	each pair, MED_IN

1.6.11 PYRAMIDS

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2,0/1)	ISCM_MAX, IFILL
cmval(id,3,0/1/.../11)	ZMIN(I), ZMAX(I), XFP(I), XBP(I), XFN(I), XBN(I), YFP(I), YBP(I), YFN(I), YBN(I),XMAX(I), YMAX(I)
cmval(id,4,0/1/2/3)	air, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,5,0/1/2/3,i)	each layer, opening, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,6,i)	each layer, opening, MED_IN
cmval(id,7,0/1/2/3,i)	each layer, surrounding, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,8,i)	each layer, surrounding, MED_IN

1.6.12 SIDETUBE

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2)	ZMIN
cmval(id,3)	ZTHICK
cmval(id,4)	ZCYL
cmval(id,5,0/1)	XMIN, XMAX
cmval(id,6)	N, Number of coaxial cylinders.
cmval(id,7,i)	each cylinder, R(I)
cmval(id,8,0/1/2/3,i)	each cylinder, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,9,i)	each cylinder, MED_IN

1.6.13 MESH

cmval(id,0)	RMAX_CM
cmval(id,1)	TITLE
cmval(id,2)	ZMIN
cmval(id,3,0/1/2/3)	X_AIR_WIDTH, Y_AIR_WIDTH, WIRE_WIDTH, WIRE_THICK
cmval(id,4,0/1)	XTOTAL, YTOTAL
cmval(id,5,0/1/2/3)	inside, ECUT,PCUT,DOSE_ZONE,IR_TO_BIT
cmval(id,6,0/1/2/3)	wire, ECUT,PCUT,DOSE_ZONE,IR_TO_BIT
cmval(id,7)	wire, MED_IN

Adjusting the mesh to fit the cells

The user inputs the wire width, w, and the x and y air widths, x_a and y_a. From these the number of holes in x and y can be calculated. The problem is that if the x or y total half-width (x_T, y_T) isn't the proper size to accomodate an odd integer number of air holes it has to be resized to do so.

Adding in the y-direction gives

$$\text{half} - \text{width} = 1/2\text{hole} + 1/2\text{wire} + \text{n}(\text{hole} + \text{wire}) + 1/2\text{wire}$$

$$y_T = (n + \frac{1}{2})y_a + (n+1)w$$

$$2n = \frac{2y_T - y_a - 2w}{y_a + w}$$

where n is the number of cells in half of the y-direction, not including the central cell. $2n$ is then the number of cells in the y-direction not including the central cell. This number must be forced to be an even integer.

If it's not an integer, it is set to the next highest integer number. If this number is not even, it is reduced again to the next lowest integer. Then the total y-width of the mesh is

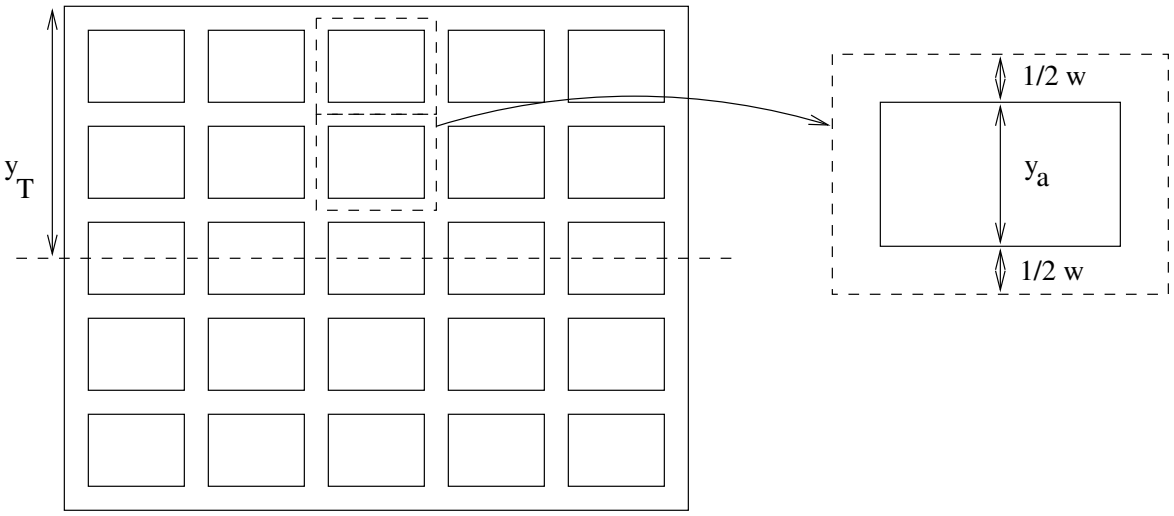

Figure 1.2: Diagram of a MESH.

reset to accomodate this number of cells (+1, the central one) of the widths specified:

$$y_T = \frac{1}{2}y_a + w + n(y_a + w)$$

1.6.14 MIRROR

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2,0/1)	ZMIN, ZTHICK
cmval(id,3,0/1)	XFMIN, XBMIN
cmval(id,4)	N, Number of layers
cmval(id,5,i)	DTHICK(I)
cmval(id,6,0/1/2/3,i)	eack layer, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,7,i)	each layer, MED_IN
cmval(id,8,0/1/2/3)	behind mirror, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,9)	behind mirror, MED_IN
cmval(id,10,0/1/2/3)	front of mirror, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,11)	front of mirror, MED_IN

Finding the angle between the z-axis and the mirror

In the inputs, d_i is the thickness of layer i. To draw the mirror we need to find the x-width of the layer. This is given by

$$\cos\alpha = \frac{d_i}{x_i}$$

or

$$x_i = \frac{d_i}{\cos\alpha}$$

Where the angle α is as shown in figure 1.3, the angle between the z-axis and the parallel lines of the layers in the mirror.

The angle α is as shown in the right triangle in figure 1.3,

$$\tan\alpha = \frac{|x_f - x_b|}{z_T}$$

$$\alpha = \tan^{-1}\left(\frac{|x_f - x_b|}{z_T}\right)$$

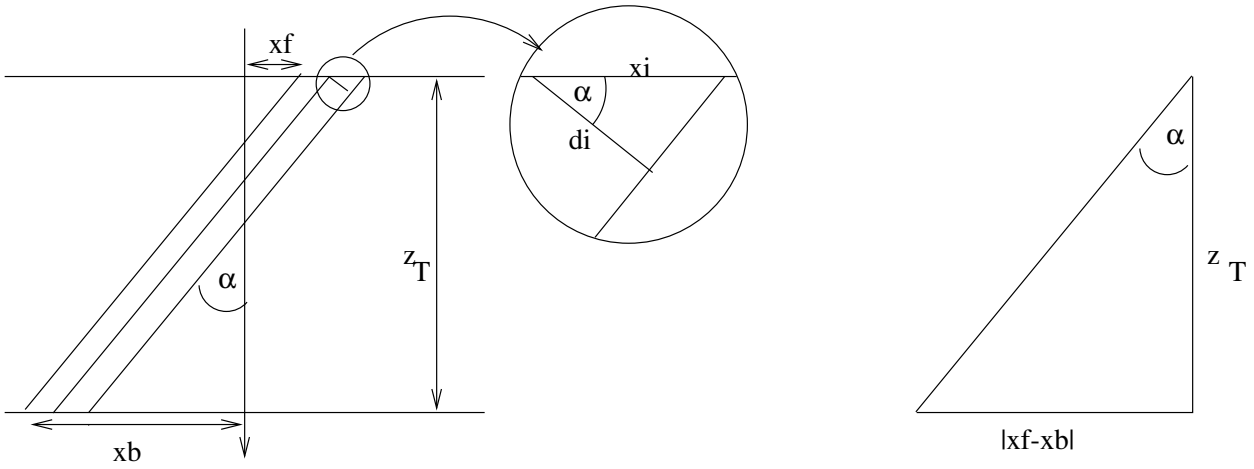

Figure 1.3: Diagram of a mirror. d_i, x_b, x_f and z_T are user-defined.

1.6.15 MLC

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2)	IDMLFC
cmval(id,3)	ZMIN
cmval(id,4)	ZTHICK
cmval(id,5)	NUM_LEAF, TWIDTH
cmval(id,6)	ZFOCUS(1)
cmval(id,7)	ZFOCUS(2)
cmval(id,8,0/1/2,i)	NEG, POS, NUM for group i
cmval(id,9,0/1/2/3)	inside, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,10)	inside, MED_IN
cmval(id,11,0/1/2/3)	leaves, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,12)	leaves, MED_IN
ngroups	number of groups

Finding the x-coordinate at the bottom

This CM has a focal point for the sides and for the ends of the collimator leaves. The user defines the coordinates at the top of the CM, so to draw the sides and ends the coordinates

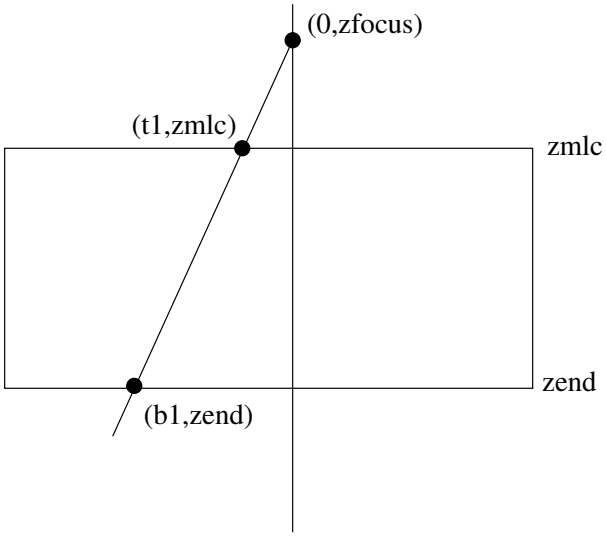

Figure 1.4: Diagram of a MLC.

at the bottom of the CM must be calculated. Simply draw a line from the focal point on the z-axis (0,$zfocus$) through the point at the top (t_1,$zmlc$) to get the equation in x and z

$$z = mx + b$$

$$zfocus = b$$

$$zmlc = mt_1 + zfocus \rightarrow m = \frac{zmlc - zfocus}{t_1}$$

We want to get the point at the bottom, (b_1,$zend$)

$$zend = \frac{zmlc - zfocus}{t_1} b_1 + zfocus$$

or

$$b_1 = t_1 \frac{zend - zfocus}{zmlc - zfocus}$$

1.6.16 XTUBE

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2,0/1)	ZMIN, ZTHICK
cmval(id,3)	ANGLEI
cmval(id,4)	N, Number of layers in the target
cmval(id,5,i)	each layer, DTHICK(I)
cmval(id,6,0/1/2/3,i)	each layer, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,7,i)	each layer, MED_IN
cmval(id,8,0/1/2/3)	front of target, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,9,0/1/2/3)	target holder, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,10)	target holder, MED_IN

Finding the x-coordinate given the thickness of the layer

The thickness of each layer is defined in the inputs, but we need the x-distance between layers in order to draw them. From the diagram, $\cos\alpha = d_i/x_i$, where d_i is the user-input thickness of the layer and α is the user-input angle between the target and z-axis. To draw the first layer, *eg.*, we need the four points xtop1, xbot1, xtop2 and xbot2. The total x-width at the top and bottom of all layers is

$$\sum_i x_i = \sum_i \frac{d_i}{\cos\alpha}$$

The top and bottom x-distances of the last layer are given by $x_0 = \frac{z_T}{2}\tan\alpha$. Then we have

$$xtop1 \quad = \quad x_0 - \sum_i x_i$$

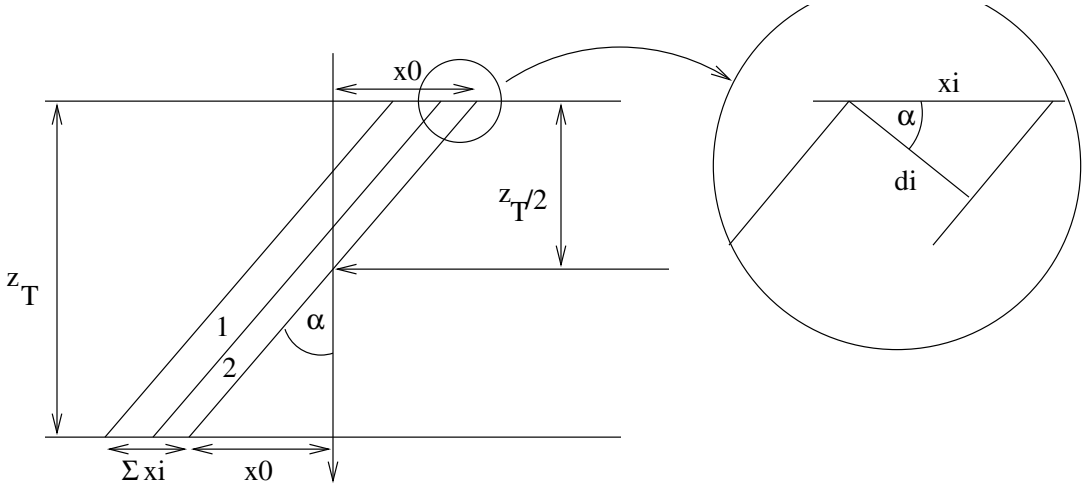

Figure 1.5: Diagram of an xtube.

$$
\begin{aligned}
xtop2 &= x_0 + x_1 - \sum_i x_i \\
xbot1 &= -x_0 - \sum_i x_i \\
xbot2 &= -x_0 + x_1 - \sum_i x_i
\end{aligned}
$$

defining the polygons of the xtube layers.

1.6.17 SLABS

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2)	N, Number of planar slabs
cmval(id,3)	ZMIN
cmval(id,4,0/1/.../6,i)	each layer, ZTHICK, ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT, ESAVEIN, MED_IN

1.6.18 ARCCHM

cmval(id,0)	RMAX_CM(ICM)
cmval(id,1)	TITLE
cmval(id,2)	ZSRC
cmval(id,3)	ZRAD1
cmval(id,4)	NUMCHM
cmval(id,5)	WIDTHCHM
cmval(id,6)	WIDTHSEP
cmval(id,7)	ARCTHICK
cmval(id,8)	FRONTHCK
cmval(id,9)	BACKTHCK
cmval(id,10)	WIDXWALL
cmval(id,11,0/1)	XMIN1, XMAX2
cmval(id,12)	ZMAX
cmval(id,13,j,0/1/2/3)	each layer j (see manual), ECUT, PCUT, DOSE_ZONE, IREGION_TO_BIT
cmval(id,14,j)	each layer j, MED_IN

Chapter 2

DOSXYZ GUI

2.1 Introduction

Currently the working source code for the GUI is stored in `$OMEGA_HOME/progs/gui`. It is written in Tcl/Tk. On this system, the versions available are Tk version 4.1 and Tcl version 7.5 with wish version 4.1 or wishx. Use wishx if you want compatibility with the SGIs here since they use an older version of wish which doesn't run the script properly. The use of older versions of tcl and tk are good for portability.

At this time, the GUI is not portable to Windows or Mac because when directory trees are specified I did not always use the 'file join' command but simply put in forward slashes, '/'.

2.2 Tcl/Tk Files for the GUI

The files required for the GUI are listed in this section. Figure 2.1 shows a flow representation of how the procedures interact with one another.

1. xyz_gui is the main script for the DOSXYZ GUI. It pops up the main window from which DOSXYZ input files may be loaded and/or edited and/or saved.

2. xyz_parameters.tcl is a file containing parameters used in the GUI. For each variable the help text is set and a default value is assigned if required (*i.e.* for an option menu). It is read only once when the GUI is started.

3. browser.tcl contains code for a directory browser. It is the same as query_filename.tcl, but it does not use a filter.

   (a) proc browse { next_proc_name dir } is the browser code.

   (b) proc set_spec_file { } sets spec_file to what the user has selected in the textbox of the browser.

4. create_file.tcl contains the code to write the inputs to a file.

   (a) proc create_file { file } writes the inputs to <file>.

   (b) proc error_flag { text } puts an error message on the screen if one of the required inputs is missing.

5. define_phantom.tcl contains code to define the phantom, either with CT data or voxel-by-voxel. These procedures are called from the "Define phantom" button on the main input window.

   (a) proc define_phantom { } puts up the proper window for data input, depending on which method was selected on the main inputs window. If it is voxel-by-voxel, it puts up a window with buttons for defining materials, x/y/z-voxel geometry as individually or groups, izscan, and the media of the voxels. If it is CT data, it puts up a window with entry boxes for the filename to use (from which it reads the media used) and other variables.

   (b) proc define_nmed { } puts up a window for the definition of media and estepe for each medium.

   (c) proc define_voxels { dir } pops up a window for the definition, in groups or individually, of voxels in direction $dir (x, y, or z).

   (d) proc voxel_med { } puts up the window to assign media to the voxels, in groups (from x/y/z, to x/y/z)

   (e) proc add_group { w } adds a group of voxels to the voxel_med window.

   (f) proc define_izscan { } puts up a window to set the output to a z-scan per page or an x-scan per page for groups of voxels, as above.

   (g) proc add_scan_group { w } adds a new group (empty entries) to the izscan window.

6. help.tcl contains 2 small procedures that act as stepping-stones to the help_dialog and help_gif routines in misc.tcl.

   (a) proc help { i } is for help that does not require a picture.

   (b) proc help_srcopts { w iopt } is for help that needs a gif file (*i.e.* source help).

7. load_input2.tcl contains procedures for loading a previous input file.

   (a) proc load_input { } calls query_filename to search for the file to load.

   (b) proc set_inp_file { tree filen } sets the variable inp_file to that selected, then calls read_input.

   (c) proc read_input { } opens the file and reads in the input parameters.

8. misc.tcl contains two help dialog box routines and tk_dialog.

   (a) proc help_dialog { w title text bitmap default args } is for help that does not require a picture. A text box with a scrollbar displays the text associated with the variable.

(b) proc help_gif { w text iconfile } is for help that requires a gif file (*i.e.* source help). The picture appears to the left and the text with scrollbar to the right.

(c) proc tk_dialog { w title text bitmap default args } is included here to change the font and the width of the message box from the default (it overrides the builtin function).

9. query_filename.tcl contains the code for browsing the directory tree.

   (a) proc query_filename { next_proc_name dir ext } pops up a window with next_proc_name as the next procedure, dir as the starting point in the directory tree and ext as the filter.

   (b) proc call_next_proc { next_proc_name } calls the next procedure.

   (c) proc update_phantfile { tree filename } set the variable PhantFileName when it is this variable being set.

10. save_input.tcl contians code for saving the input file.

    (a) proc save_input { } calls query_filename to get the name of the file to save it as.

    (b) proc DoesInpFileExist { tree filename } checks whether this file exists and if it does, prompts the user with "Are you sure?" (overwrite_prompt).

    (c) proc save_inp_file { tree filen } sets the inp_file variable to its new value then calls create_file to write it.

    (d) proc overwrite_prompt { tree filename } puts up a dialog box if the file selected already exists.

11. set_main_inputs.tcl contains code for the main input window.

    (a) proc edit_parameters { } pops up the main window and places the main input parameter text boxes or option menus on it, as well as the radiobuttons for how to define the voxels and the button to do it, and the option menu for sources, which pops up the source option window automatically.

    (b) proc set_value { io iopt w } set the value of a variable with an option menu.

    (c) proc set_presta { } pops up a window to set the PRESTA inputs if the default is not used.

12. set_source.tcl contains code for setting the source options.

    (a) proc set_src_options { iopt } pops up the main source input window, depending on which source was chosen.

    (b) proc enable { flag } enables or disables buttons/menus/textboxes depending on which source is chosen.

    (c) proc set_inc_or_exc { parent index } pops up a window in which the user chooses LATCH bits for inclusive and/or exclusive filters.

    (d) proc close_latbit { index } saves the LATCH bit settings selected in the variable latbit when the previous window is closed.

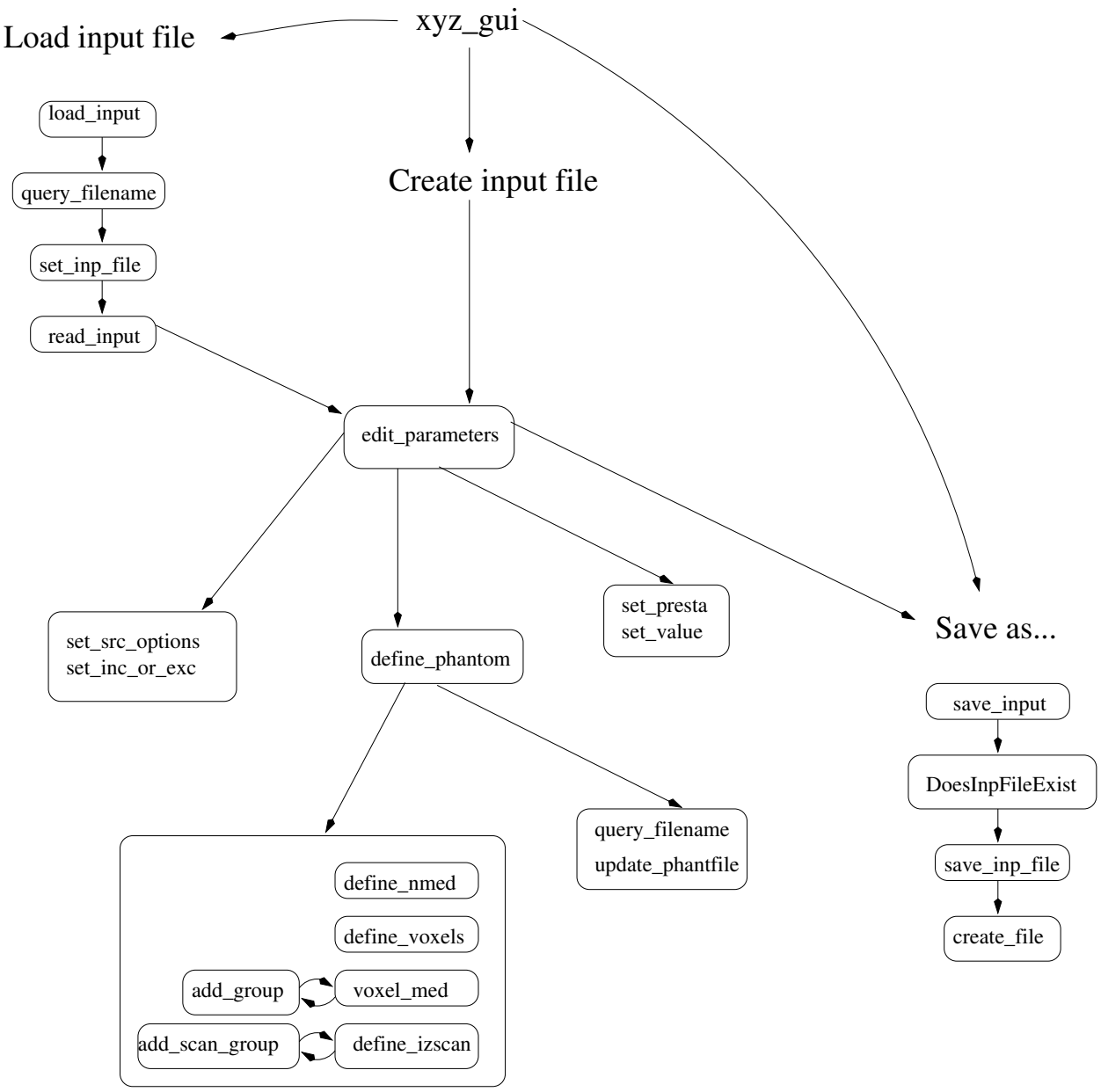

Figure 2.1: A flow diagram of how the procedures interact. Note that not all are presented (those that are not directly linked to another).

2.3 Naming Conventions

2.3.1 Main

Name	Meaning
values(1)	title
values(2)	nmed
CTdataflag	flag for ct/non-ct phantom
values(5)	ncase
values(6)	iwatch
values(7)	timmax
values(8)	inseed1
values(9)	inseed2
values(10)	beam_size
values(11)	ismooth
values(12)	irestart
values(13)	idat
values(14)	ireject
values(15)	esave_global
values(16)	presta inputs default/user set
values(16,0/1/2/3/4)	presta inputs
values(20)	zeroairdose
values(21)	doseprint
values(22)	max20

2.3.2 Source

Name	Meaning
values(3)	electron, photon, positron option menu
values(4)	source number
numsrcopts(i)	for each source i, is the number of source options required.
srcoptnames(i,j)	for each source i, is the label used for source option j.
values(4,j)	value of source option j for the source selected.
Ein	energy of monoenergetic beam.
spec_file	spectrum file for sources.
enflag	flag for the sources (ENFLAG)
medsur	medium surrounding phantom (MEDSUR)
dsurround	thickness of region surrounding phantom (DSURROUND).
values(17)	mode option menu (phase space file format)
values(18)	i_bit_filter
nbit1	number of bits to be used for the filter
nbit2	number of bits to be used for the second half of the filter (incluse and exclusive) (nbit1+nbit2 ¡ 29)
inc	is an array of length 29 for inclulsive filter; if bit j is on, inc(j)=1, else inc(j)=0
exc	is an array of length 29 for exclusive filter; if bit j is on, exc(j)=1, else exc(j)=0
latbit(i)	holds the bits to include/exclude (i=1,nbit1 and i=nbit1+1,nbit2)

2.3.3 Define media

Name	Meaning
imax(0/1/2)	Number of voxels in X/Y/Z
medium(i)	Name of medium i
ecutin	electron cutoff energy
pcutin	photon cutoff energy
estepm(i)	estepe for medium i
smax	maximum step size, default 5cm
PhantFileName	filename to be used for pahntom created by CT data.

2.3.4 Define voxel geometries

If imax(0)>0, define individually. Input x0,x1,x2,...,xN (likewise for y and z):

Name	Meaning
ivox(0,i)	xi, x-coord of end of voxel (x(i-1) is the x-coord of start of voxel)
ivox(1,i)	yi, y-coord of end of voxel (y(i-1) is the y-coord of start of voxel)
ivox(2,i)	zi, z-coord of end of voxel (z(i-1) is the z-coord of start of voxel)

If imax(0/1/2)<0, input min,width,number of voxels:

Name	Meaning
gvox(0,0)	min x for group i
gvox(0,i,1)	x-width for group i
gvox(0,i,2)	number of voxels with this width in group i
gvox(1,0)	min y for group i
gvox(1,i,1)	y-width for group i
gvox(1,i,2)	number of voxels with this width in group i
gvox(2,0)	min z for group i
gvox(2,i,1)	z-width for group i
gvox(2,i,2)	number of voxels with this width in group i

2.3.5 Define voxel media

Define material and density and IZSCAN in groups:

Name	Meaning
nrow	Number of material groups
izrow	Number of iz groups

For group i (i=1,nrow):

Name	Meaning
mvox(i,0,0)	x from
mvox(i,0,1)	x to
mvox(i,1,0)	y from
mvox(i,1,1)	y to
mvox(i,2,0)	z from
mvox(i,2,1)	z to
mvox(i,0)	material
mvox(i,1)	density

For group i (i=1,izrow):

Name	Meaning
izvox(i,0,0)	x from
izvox(i,0,1)	x to
izvox(i,1,0)	y from
izvox(i,1,1)	y to
izvox(i,2,0)	z from
izvox(i,2,1)	z to
izvox(i)	IZSCAN for group i