Create Applications with

Geeonx

V. 31. July 2022

© by Rasmus J. N. Keller

- 1 -

Basics

1. Before all

Download GeeonxCreator together with the shared library, GeeonxDemo and the sourcecode of GeeonxDemo at geeonx.de. Follow the install instructions.

2. Geeonx objects

Everything is about Geeonx objects. They transport all content data of a Geeonx application. Geeonx objects are described and defined in the structure **struct geeonx_object**. Geeonx objects are omnipotent. By changing the parameter **uint32_t obj_status** they can alter their character.

Let's take a look after the different types of Geeonx objects:

obj_status:

1 = window
2 = button
3 = big_text_object
4 = image
5 = small_object or input_form
6 = small_object or input_form for numbers only
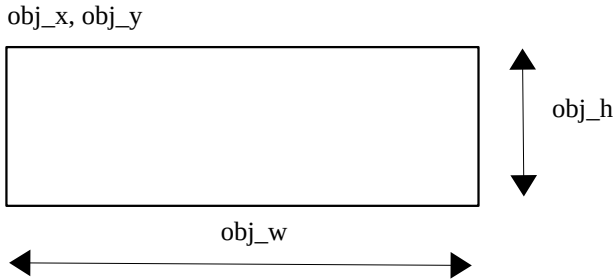**7 = menu**
8 = connected object
9 = internal use: image loaded
10 = bitmap

Before we have a closer look to the different types of Geeonx objects we want to learn to handle the parameter that all have in common.

3. Size and colour of Geeonx objects

All Geeonx objects are rectangles. Therefore we have got the typical x, y and w, h parameter.

obj_x, obj_y

obj_h

obj_w

Geeonx objects have a colored background. The color information is stored in the object 0-255 rgb variables:

uint32_t background_color_r;
uint32_t background_color_g;
uint32_t background_color_b;

4. Mothers and daughters

There are independent objects and dependent objects. The first ones are earmarked with **uint32_t mother_object=0.** Dependent objects are linked via **uint32_t mother_object** with their mother. In addition the dependent objects are stored in **uint32_t move_with[100]** of their mother. Hence each dependent object can be addressed from his mother with move_with [0-99]. Windows and pull-down-menus are independent objects whether buttons and pictures are typically dependent objects.

The independent objects like windows can be switched on or off with the struct entry **uint32_t disabled**. If you set disabled to 1, Geeonx library won't draw the disabled object in the next redraw case. The dependent objects will be drawn if they are at least to some extent inside the **text space** of the corresponding mother_object. Vice versa they will be disabled and therefore not drawn if thy leave the text space. Hence, there is no need to switch them off (see 5.).

Important: If **out_of_textborders** is activated in regard to the depending object,

3

the part of the depending object might also be in the side area of the mother_object to remain or get activated by the library.
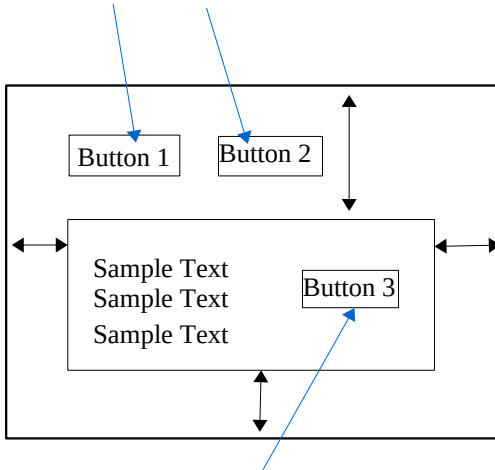
5. Text space

Every Geeonx object with exception of image and bitmap can contain text. The text is normally left formatted. Buttons are center formatted. You can set the text distances with the following Geeonx object variables:

uint32_t text_distance_left;
uint32_t text_distance_right;
uint32_t text_distance_top;
uint32_t text_distance_bottom;

The inner rectangle build by the text_distances is also the place for **depending objects**. They are moved together with the text. To place depending objects outside the inner space set the value **uint32_t out_of_textborders** to one.

Important: This will also release the objects from being moved !

Buttons with **out_of_textborders=1** – will not be moved !

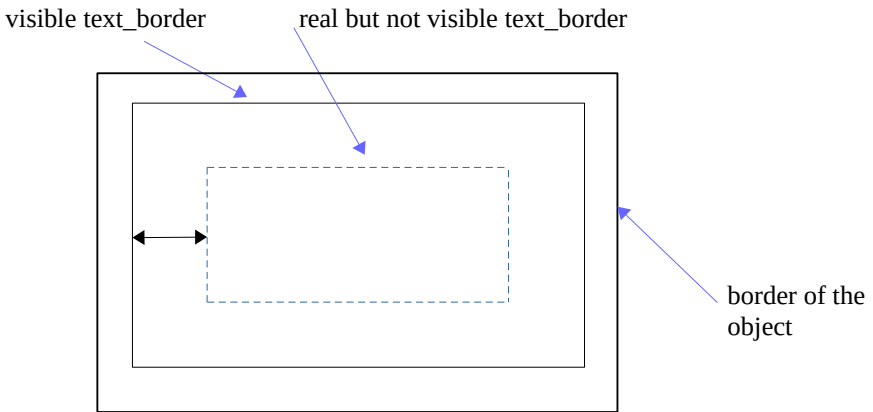Button with **out_of_textborders=0**

⟵⟶ Text distances

6. Visual effects

With **uint32_t visual_effect** you can activate some visual effects. By setting visual_effect to 1 you will activate a visible text_border line.

You can select the color of the visual effect with the following rgb variables:

uint32_t border_color_r;
uint32_t border_color_g;
uint32_t border_color_b;

The distance of the visible border line to the real text borders can be adjusted with **uint32_t border_distance**.

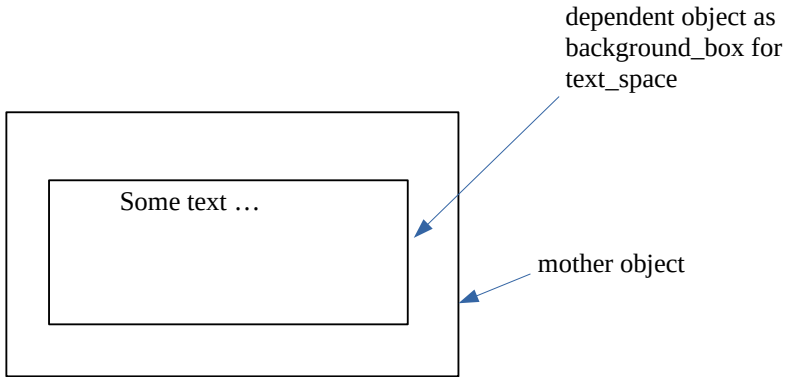visible text_border   real but not visible text_border

border of the object

◄─► border_distance

With **uint32_t visual_effect** you can take the following decisions:
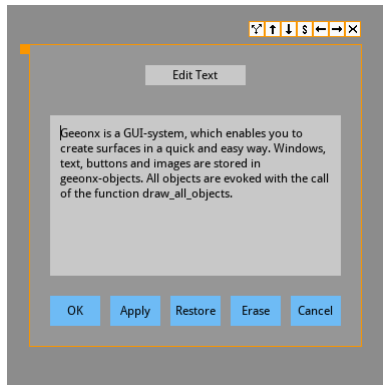
1 – visible text_border

2 – background_box

3 – visible text_border + background_box

4 – visible border

5 – visible border + background_box
Background_box is a special feature for dependent objects. If you have activated

background_box, the Geeonx library will take the outlines of the text_space of the corresponding mother_object as x,y,w,h value for the object with activated feature. Deviating from the general rule Geeonx will take the colour values of the normal **background_color_r,g,b** values. You must also switch **out_of_textborder** to 1 to avoid the scrolling of the background_box.

dependent object as background_box for text_space

Some text …

mother object

Do have editable text in the background_box the dependent object must have obj_status=3 and cursor_status=1. Without doing nothing you have an window with text editing functionality for the **struct geeonx_win_stream** (see next page) element **uint8_t object_text[1000]**. Like this:

The visible_border effect simply outlines the object.

7. Text

Text of the so called big_objects is stored outside the Geeonx objects. Big_objects are the following obj_status types:

1 - window
3 - big text_objects
8 - connected or chain_objects

It is stored in the so called win_streams:

struct geeonx_win_stream
{
      uint32_t object_number;
      uint32_t position_in_stream;
      uint8_t object_text[1000];
};

Streams are mutually linked with the connected Geeonx object. The number of the actual connected stream is stored in the struct geeonx_object variable **uint32_t stream_identifier.** Vice versa the number of the connected object is stored in **uint32_t object_number.**

The linkage between the big_object and the stream is managed by GeeonxCreator automatically during the design process of your interface. Hence, it is never needed to know the value of stream_identifier. It is even better not to know the value because it is dynamically adapted by GeeonxCreator.

> **Important:**
> To establish the linkage it is necessary to activate it with GeeonxCreator by selecting the entry 'Edit Text' within the menu 'Text'. After clicking 'OK' to close the window and finish the activation process the linkage is enabled. You can convince yourself by reading the value of stream_identifier within the 'Text Parameters' dialog.

Text of the so called small_objects is stored inside the Geeonx objects. Small_objects are the following obj_status types:

2 - button
5 - small_object or input form
6 - small_object or input form for numbers only

The text of small_object is stored in the element **uint8_t obj_text[65].**

- 2 -

Launch SDL2 libraries
and
Settings

Let' s start SDL:


```
if(SDL_Init(SDL_INIT_VIDEO|SDL_INIT_JOYSTICK) !=0)
{
     printf("SDL can not be initialized: %s\n", SDL_GetError());

}
```


Start SDL TTF for font support:

```
if(TTF_Init()==-1)
{
   printf("TTF_Init: %s\n", TTF_GetError());

}
```


Your Geeonx application is maintained with the **struct geeonx_appdata.** First of all with geeonx_appdata all information of your Geeonx application is gathered.

In particular some SDL relevant data is stored in this place. For example the pointer to the Geeonx standard font of your application:

8

TTF_Font *font;

Let's have some settings. First you should declare a **struct geeonx_appdata.** I normally use a pointer called myapp.

struct geeonx_appdata *myapp;

I recommend to reserve some memory from the heap:

```
/* Get memory ... */
```

myapp=(struct geeonx_appdata *)calloc(1, sizeof(struct geeonx_appdata));

After that you have got a nice pointer to work with.

First decide the resolution you want to use.

For example choose:

```
myapp->screen_w=1024;
myapp->screen_h=600;
```

Please tell Geeonx also the coordinates of your window rectangle:

```
myapp->hole_screen.x=0;
myapp->hole_screen.y=0;
myapp->hole_screen.w=1024;
myapp->hole_screen.h=600;
```

You have to do some SDL2 settings and provide Geeonx with the necessary information:

Define a SDL2 window for your application:

```
myapp->geeonx_window = SDL_CreateWindow("Geeonx Demo",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
myapp->screen_w, myapp->screen_h, SDL_WINDOW_SHOWN);
```

Furthermore a SDL2 32 bit color surface is required:

myapp->screen= SDL_CreateRGBSurface(0, myapp->screen_w, myapp-

>screen_h, 32, 0x00FF0000, 0x0000FF00, 0x000000FF, 0xFF000000);

After that connect your SDL2 window (your Geeonx screen) with renderer:

myapp->renderer = SDL_CreateRenderer(myapp->geeonx_window, -1, 0);

Make the rendering smoother:

SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "linear");

Create a texture out of your bitmap surface:

myapp->texture=SDL_CreateTexture(myapp->renderer,
SDL_PIXELFORMAT_ARGB8888, SDL_TEXTUREACCESS_STREAMING,
myapp->screen_w, myapp-> screen_h);

That's all SDL2 stuff ;-) - for instance.

If you don't use a picture with size of your Geeonx screen as background, Geeonx
will always redraw the background in your favorite color stored in:

myapp->background_r=50;
myapp->background_g=50;
myapp->background_b=50;

You have also to choose the colour of the outlines of a selected_window and the
little selected_box for the case of size_mode is activated or scroll_mode is
activated:

myapp->size_r=250;
myapp->size_g=150;
myapp->size_b=0;

myapp->scroll_r=110;
myapp->scroll_g=187;
myapp->scroll_b=245;

You can preselect size_mode with:

myapp->switch_st_scroll_size=0; /* You can size windows. */

Or you choose 1 for scroll_mode.

You may chose German as system language for support of the German ä, ö, ü and ß:

myapp->language=1; /* Chose German language. */

You may chose insert or delete modus for text editing. Set 0 for insert modus or 1 for delete modus.

myapp->insert_modus=0;

Furthermore you can adjust the scroll_speed:

myapp->scroll_speed=3;

Important: The variable active_windows should 0 at the beginning.

myapp->active_windows=0;

- 3 -

Getting started

Geeonx Creator will store the objects in an .gee file and the streams in an .gew file with the same name. Theoretical the names can differ.

/* Start geeonx ! */

gee_start_geeonx(myapp, 0,0,"your_app.gee", "your_app.gew");

Set your standard application font:

myapp->font=TTF_OpenFont("DroidSans.ttf", 11);

It is needed to activate window operators.

gee_set_operators(myapp);

It is necessary to draw a nice background for your application.

gee_draw_box(myapp, 0, 0, myapp->screen_w,myapp->screen_h,myapp->
background_r,myapp->background_g,myapp-> background_b);

Do text formating and draw all Geeonx objects and update the screen with one
function call.:

gee_draw_all_objects(myapp, 1, 1);

Congratulations – the work before is done. You should start with a loop for the
examination of SDL events like the sample in geeonx_demo.c. It is important not
to change or erase the function calls to Geeonx library functions to keep the
Geeonx GUI working. You are allowed to use the geeonx_demo.c source code for
your own applications. Don't hesitate to use it.

- 4 -

Do something

1. Addressing

The Geeonx objects are stored in a reserved part of the heap. The address is
myapp->object_data. The first Geeonx object is stored at **myapp-
>object_data+1**.

So if you want to change the width of the Geeonx object with no. 103 to 200 px
you just do the following:

(myapp->object_data+103)->obj_w=200;

The start address of the win_streams is stored at **myapp->win_stream**. The
number of the win_stream that is connected with the big_object is stored in
(myapp->object_data+number)-> stream_identifier.

I recommend to first read the corresponding stream_identifier.

stream_identifier=(myapp->object_data+number)->stream_identifier;

Please keep in mind that stream_identifier is a dynamic value that may change during the execution of the program.

After reading of stream_identifier you can copy text into the stream like this:

gee_copy_st_last_part(&message_text[0], &(myapp-> win_stream +stream_identifier)->object_text[0]);

2. Drawing and screen update

With the function **gee_draw_all_objects** you can evoke and update all your Geeonx objects:

int gee_draw_all_objects(struct geeonx_appdata *myapp, unsigned int format, unsigned int redraw)

By setting format to 1 Geeonx will format all text of Geeonx objects. By setting redraw to one Geeonx will make a full screen update after the drawing work is done.

3. Work with windows

For serious work you will need windows. Of course you should first create and design your windows and all other Geeonx objects with the tool Geeonx Creator (see chapter 8).

If you want to use a window it is necessary to activate it before. You select the window to be activated with **myapp->in_new_object**:

myapp->in_new_object=8;

gee_activate_window(myapp);

After that you should do an update of Geeonx objects and screen.

gee_draw_all_objects(myapp,0,1); /* 1 for screen update */

To close a window use the function **void gee_close_win(struct geeonx_appdata *myapp, unsigned number).** Number is of course the number of the Geeonx window to be closed. If you choose 0 as number the actual selected_window will be closed.

4. Window update

If you don't want to update the whole screen, you can use **void gee_draw_and_screenup_selected_window(struct geeonx_app- data *myapp)** to draw and screen update only the selected_window.

gee_draw_and_screenup_selected_window(myapp);

If you want to update selected_window without making a screenupdate use **void gee_draw_selected_window(struct geeonx_appdata *myapp)**.
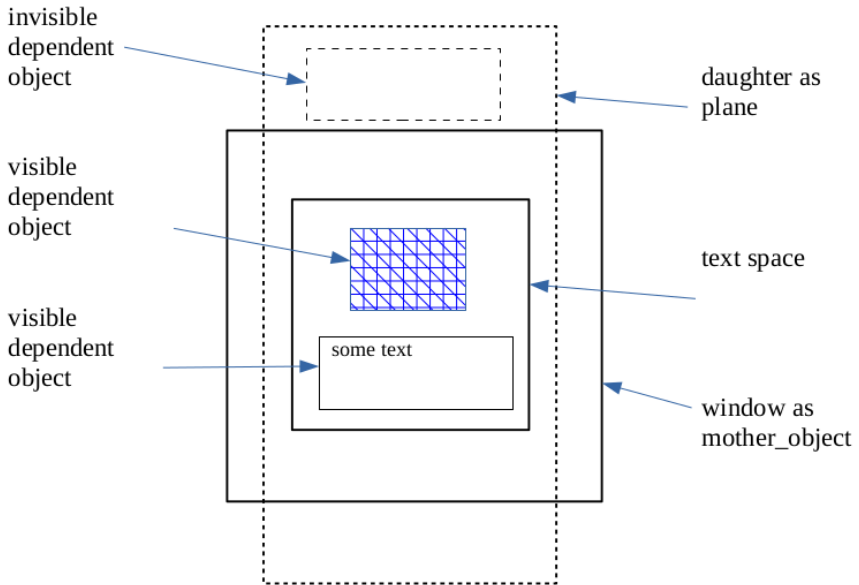
Do have an fully working text editing window just activate cursor_status=1 for struct geeonx_win_stream element uint8_t object_text[1000]. Scrolling, editing, redrawing everything is done by geeonx.

Quite easy isn't it?

5. Working with complex content in windows

For complex content it is recommendable to use a big object as sliding plane as surface for text, pictures and further content. There for take a window and a big text_object as daughter. On the daughter as sliding plane the other objects are placed as dependent objects of the daughter. The plane and the objects placed on the plane that aren't inside the text_area, are no visible.

In case of a scrolling event the sliding plane will be scrolled with all the contents automatically by the Geeonx library.

invisible
dependent
object

daughter as
plane

visible
dependent
object

text space

visible
dependent
object

some text

window as
mother_object

6. Working with larger text in windows

Sometimes we need to work with larger text volumes – perhaps for text processing. Geeonx offers the tools for that purpose.

The **struct geeonx_object has two interesting members:**

uint8_t *big_stream_address;
uint32_t position_in_big_stream;

*big_stream_address is char pointer to an text stream deposited in an allocated part of the memory. The memory is allocated and the text stream is connected to a geeonx_object (**chain_mother**) with the function:

int gee_prepare_chain_mother(struct geeonx_appdata *myapp, unsigned int number, unsigned int size)

Of course you have to know the size of your text stream.

Here is some demo code from geeonx_demo.c:

```
error=0;

error=gee_prepare_chain_mother(myapp, 32,20000);

if(error==-1)
{
        printf("Can't get enough memory for chain mother ! \n \n ");
        gee_close(myapp);
        exit(0);
}

error=0;

gee_read_external_stream_copy_to_object(myapp,"input.txt",32);


if(error==-1)
{
        printf("Can't load chain mother ! \n \n ");
        gee_close(myapp);
        exit(0);
}
```

int gee_read_external_stream_copy_to_object(struct geeonx_appdata *myapp, unsigned char filename[], unsigned int number)

will read your text stream as txt-file and copy it into the allocated memory. **position_in_big_stream** records the cursor position in your stream.
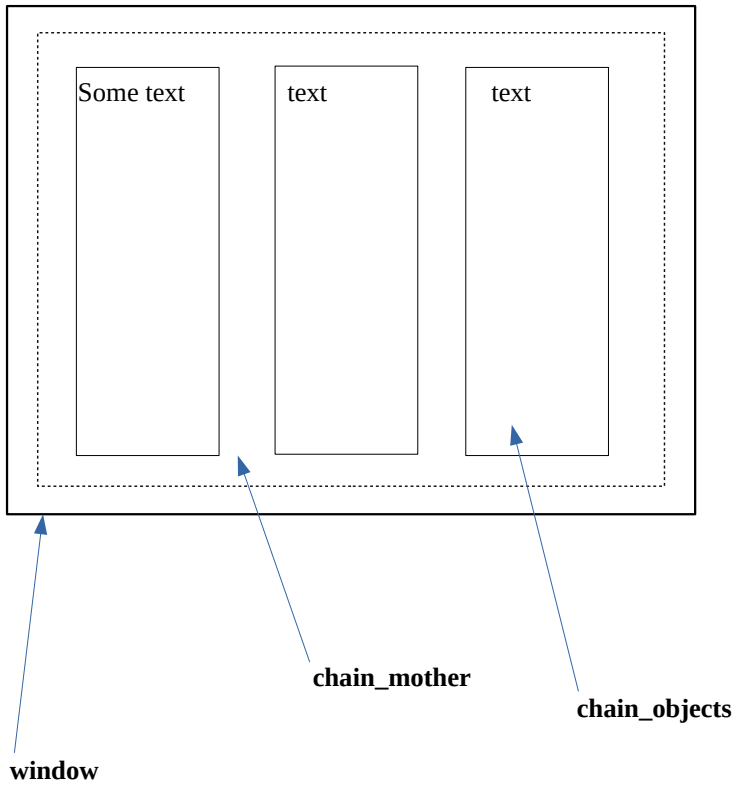
As you already have learned gee_draw_all_objects(myapp,1,1) will format and draw all objects. The big stream will be left formatted and distributed to the depending objects of the chain_mother.

For formatting you can use the following function also seperately:

void gee_format_stream_to_linked_objects(struct geeonx_appdata *myapp, unsigned int big_stream_mother)

The cursor_status of every chain_object must be set to 1.

After that Geeonx provides multi column text_processing.

Some text text text

chain_mother

chain_objects

window

7. Click events

Any click on a Geeonx_object is stored into **myapp->in_new_object**. Hence, if you want to check if an user has clicked on a button or on a menu entry you simply have to examine the content of **myapp->in_new_object**.

That's what happens in geeonx_demo.c in function **myapp_exa_left_selected_object(struct geeonx_appdata *myapp).**

For example it is checked if the "Okey" button of the "About"-Window is clicked by the user. In this case the window will be closed and a redraw and screen update will follow:

```
if(myapp->in_new_object==20)
{
        /* Close window. */

        gee_close_win(myapp,0);


        /* Do complete redraw. */
        gee_draw_all_objects(myapp,0,1);
}
```

- 5 -

Structure of a Geeonx Application

The center of a Geeonx Application is an event loop using the function SDL_PollEvent and recording the event in the geeonx application structure:

SDL_PollEvent(&myapp → event);

After that the different events are checked and processed. The SDL_Events SDL_MOUSEBUTTONDOWN, SDL_MOUSEBUTTONUP and SDL_KEYDOWN, SDL_KEYUP are examined.

In regard of mouse buttons it is necessary that Geeonx receives x and y position of the mouse to check if a Geeonx object is clicked.

myapp->in_new_object=gee_test_all_objects(myapp, myapp->event.button.x, myapp->event.button.y);

After that Geeonx must be considered to process windows events. The use of these functions is mandatory for the use of Geeonx window. For example:

gee_exa_left_released(myapp);

As next step you can fork to a function for checking the events of your application:

myapp_exa_left_selected_object(myapp);

With the use of the functions

gee_getfirstkeyinformation(myapp),
gee_getfinalkeyinformation(myapp),

gee_get_text(myapp)

Geeonx proceeds keyboard inputs.

The following functions in the central event loop are mandatory:

myapp->in_new_object=gee_test_all_objects(myapp,myapp->event.button.x, myapp→event.button.y)

gee_exa_left_pressed(myapp)

gee_select_window(myapp)

gee_exa_left_released(myapp)

gee_getfirstkeyinformation(myapp)

gee_getfinalkeyinformation(myapp)

gee_get_text(myapp)

To connect an action with and as long as a mouse button is pressed the BUTTONDOWN-event must first be buffered together with the x and y positions of the mouse.

```
case SDL_MOUSEBUTTONDOWN:


        if(myapp->event.button.button==SDL_BUTTON_LEFT)
        {

                buffer=1;

                buffer_x=myapp->event.button.x;
                buffer_y=myapp->event.button.y;
        }

        break;
```

Later on the event can be used after closing of the switch element. The function Geeonx system function gee_exa_left_pressed is therefor located after the switch element:

```
if(buffer==1) /* left plus pressed */
{

        myapp→in_new_object=gee_test_all_objects(myapp, buffer_x,
        buffer_y); /* Test all objects. */


        gee_exa_left_pressed(myapp);

        gee_select_window(myapp);

        myapp->in_new_object=0; /* Reset in. */
}
```

For further understanding take a look at the geeonx_demo.c source.

Communicate with the user

1. Short message dialog without scrolling

If you want to have dialog window without scrolling, you should activate the **fix_dialog** functionality by choosing the se tting 1.

(myapp->object_data+your_object)->fix_dialog=1;

In regard to such fixed dialogs Geeonx already support center formatting of text. For this feature you should set **text_align** to1.

(myapp->object_data+your_object)text_align=1;

2. Force an user answer

Of course you can check clicks on window buttons by normal event checking. But sometimes your program really need answer to move on. Therefor you should use the functions

unsigned int gee_get_mono_mouse_selection(struct geeonx_appdata *myapp, unsigned int button);

or

unsigned int gee_get_mouse_selection(struct geeonx_appdata *myapp, unsigned int one, unsigned int two);

The difference between both functions is that mono_mouse is for dialogs with one button and mouse_selection for dialogs with two buttons.

button=gee_get_mouse_selection(myapp, 141, 142);

```
if(button==141)
{
        /* Do something. */
}
else if(button==142)
```

```
{
        /* Do something else. */
}
```

3. Loading content to windows

With **gee_load_window_text** you can easily update window content. The content is copied into the stream and after that the text will be formatted. If you want tp format additionally two buttons use **gee_load_window_text_buttons.**

void gee_load_window_text(struct geeonx_appdata *myapp,unsigned int window_number, unsigned char *main_content_p)

void gee_load_window_text_buttons(struct geeonx_appdata *myapp,unsigned int window_number,unsigned char *main_content_p, unsigned int button_one, unsigned int button_two)

strcpy(&content[0],"Can't open gee-file. Try again or quit ?");

strcpy(&(myapp->object_data+141)->obj_text[0],"Retry");

strcpy(&(myapp->object_data+142)->obj_text[0],"Quit");

gee_load_window_text_buttons(myapp, 140,&content[0],141,142);

4. Simple user notification

For a simple user notification with a window dialog use

void gee_notice_window(struct geeonx_appdata *myapp, unsigned int number, unsigned int mono_button).

Number is the number of the window. Mono_button is the number of the button to be clicked by the user. Geeonx will draw the dialog and wait for the user to click the button. After that Geeonx will erase the dialog window. For you remains to do a redraw.

gee_notice_window(myapp, 406, 409);

gee_draw_all_objects(myapp, 0, 1);

5. Menus

First of all you should design an Geeonx menu object as container. Define the outlines and chose obj_status=7. Disable the object as default. Define the menu entries as dependent objects.

With the function

int gee_move_out_menu(struct geeonx_appdata *myapp, unsigned int number, unsigned int function)

you can use your created menu. The function enables and draws the menu. It has got its own event checking functionality. If the user moves with mouse out of the menu, it closes itself. The selection (clicked entry) will be returned as unsigned int value. In case of selection or users moves out of the menu, the menu object erases and disables itself. You only should do redraw afterwards. Of course you can do some other drawing stuff before doing a redraw and screen update.

For a pull-down-menu set parameter function to 1. If you want to have a pop-up-menu chose 0.

```
selection=gee_move_out_menu(myapp, 31, 0);


if(selection==32)
{
        /* Do something. */
        /* Redraw and screen update. */

        gee_draw_all_objects(myapp,0,1);
}
else if(selection==33)
{
        /* Do something else. */
        /* Redraw and screen update. */

        gee_draw_all_objects(myapp,0,1);
}
else
{

        /* Redraw and screen update. */
```

```
            gee_draw_all_objects(myapp,0,1);
}
```

6. Input forms

First create a Geeonx form object. This could be a small object with status 5 or a small object with status 6 for input forms for numbers only.

You must set Geeonx object variable **cursor_input_range** to a value higher than 0 but smaller than 65. Cursor_input_range represents the number of letters that can be inserted at maximum.

You must also switch the Geeonx variable **cursor_status** to 1.
If you make a big_object editable by also setting cursor_status to 1 and give cursor_input_range a value larger than 0 and smaller than 1000.

That's it. Geeonx will do the rest. Of course you find the user input in the corresponding obj_text (small_object) or in the corresponding win_stream (big_object).

After changing the input text by the program (not by the user) it is necessary to update the internal cursor and line data. Please use for this purpose or for any other reset:

void gee_reset_textobject(struct geeonx_appdata *myapp, unsigned int big_object, unsigned int number)

For small input forms set big_object=0 for big_objects set big_object=1.

7. Getting numbers out of char forms and vice versa

Use **void gee_transform_str_to_num(unsigned char *string,unsigned int *number)**

to make an unsigend int out the char input of your form. For example your form has got the number 225 and you want x to receive the unsigned int value:

gee_transform_str_to_num(&(myapp->object_data+225)->obj_text[0], &x);

Vice versa:

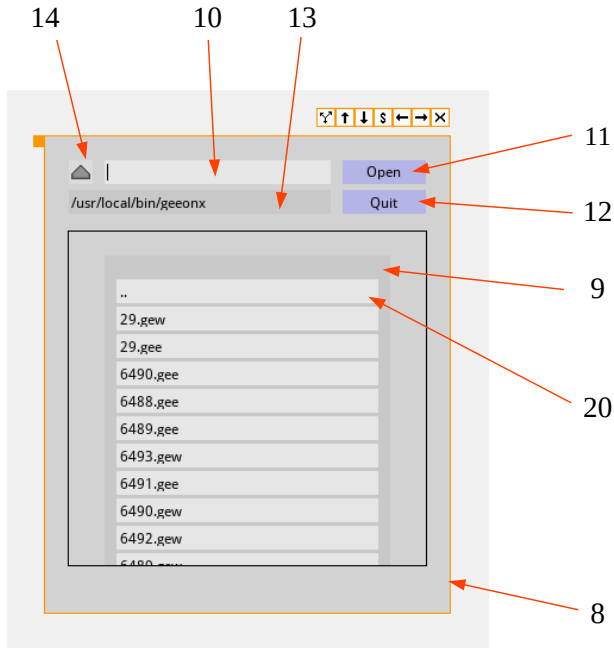**gee_transform_num_to_str(&x,&(myapp->object_data+225)->obj_text[0]);**

int gee_transform_num_to_str(unsigned int *number, unsigned char *string)

Function will return -1 if the number is higher than 1000000000.

- 7 -

Geeonx fileselector

Take a look at the Geeonx fileselector:

If you use the template "template_wselector.gee" the objects 8-14 and 20 are used for the fileselector. 8 is the fileselector-window, 10 is used for the input form and 11, 12 for the buttons. With number 4 a "directory up" button was added. Number 9 is used for the darker glider. 20 is daughter of 9. It is used as first entry of the directory list and template for the others. All entries are stored in daughters of 9

beginning with (myapp->object_data+9)->move_with[0]=20. Hence at maximum 100 [0-99] directory entries are supported within this system.

Before using the fileselector Geeonx must once design all entries according to the template object 20. Therefor you must tell Geeonx the number of the last daughter to b used as entry.

void gee_prepare_fileselector(struct geeonx_appdata *myapp, unsigned int number, unsigned int last_move_with)

gee_prepare_fileselector(myapp, 9, 99);

Here the dependent objects 0-99 are used as directory entries. Geeonx will use the object stored in move_with[0] as template and will use the following numbers as space for further entries. In our example 20 is the first entry. Geeonx will use the following objects as entries [20, 21 ….. 119].

Before each use of the fileselector you should read the actual directory and copy it into the entries of the selector. Geeonx offers for this purpose the functions:

void gee_read_app_dir(struct geeonx_appdata *myapp)

void gee_load_dir_entries(struct geeonx_appdata *myapp, unsigned int number)

Number is the mother_object of the entry objects.

For example:

gee_read_app_dir(myapp);

gee_load_dir_entries(myapp,9);

Here is an example to realize the selection of the entries by mouse click within the normal click examination:

/* Check if fileselector is selected_window. */

if(myapp->selected_window==8)
{
    counter=0;

```
    while(counter<=99 && finished==0)
    {
        depending_object=(myapp->object_data+9)->
    move_with[counter];

        if(myapp->in_new_object==depending_object)
        {
                finished=1; /* File selected */
        }
        counter++;
    }

    /* Copy text of selected_object to fbox of filename.  */

    if(finished==1)
    {
        strcpy((myapp->object_data+10)->obj_text,(myapp->
    object_data+depending_object)->obj_text);

        gee_reset_textobject(myapp,0,10);

        gee_draw_and_screenup_selected_window(myapp);
    }

    if(myapp->in_new_object==11) /* Load or Save! */
    {
        /* Do some load or save action. */
    }
}
```

- 8 -
- Primitives and Bitmaps

Geeonx offers the possibility to draw primitives like pixels, lines and boxes on the screen. The color values are stored in unsigned char r, g, b. The value can be 0-255.

void gee_draw_pixel(struct geeonx_appdata *myapp, unsigned int x, unsigned int y, unsigned char r, unsigned char g, unsigned char b)

void gee_draw_line(struct geeonx_appdata *myapp, unsigned int fx, unsigned int fy, unsigned int lx, unsigned int ly, unsigned char r, unsigned char g, unsigned char b)

void gee_draw_box(struct geeonx_appdata *myapp, unsigned int x, unsigned int y, unsigned int w, unsigned int h, unsigned char r, unsigned char g, unsigned char b)

To draw a line we need a first point fx, fy and a last point lx, ly. In regard to pixels and lines it is needed to lock the SDL surface before drawing and afterwards unlock the surface. In case of boxes the lock and unlock process is done by the library.

gee_lock(myapp->screen);

draw something ….

gee_unlock(myapp->screen);

> Please note that the functions haven't got a clipping functionality. Hence, drawing over the borders will provoke a memory violation !

Maybe you want to draw something in a window and not on the screen. For this purpose Geeonx provides bitmap objects. Do you remember? Bitmap objects are Geeonx objects with obj_status=10.

For drawing in a bitmap object a SDL2 32 bit color surface is required. Address is stored in **(myapp->object_data+number)->image_surface.** The width and height of the object is used for width and height of the surface:

(myapp->object_data+46)->image_surface=
SDL_CreateRGBSurface(0, (myapp->object_data+46)->obj_w, (myapp->object_data+46)->obj_h, 32,rmask, gmask, bmask, amask);
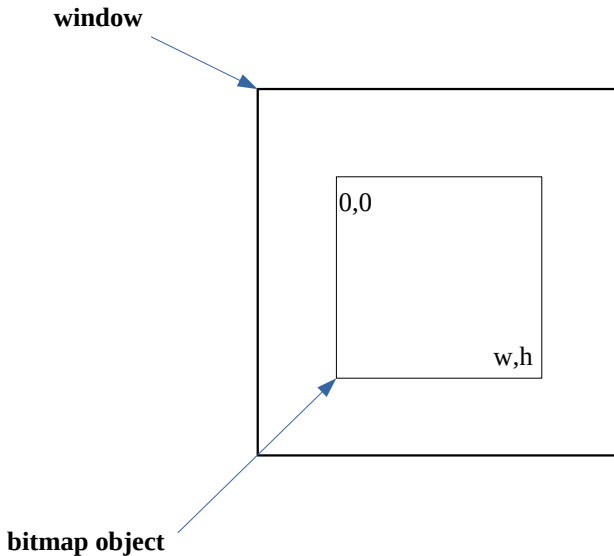After creating the surface connected to a bitmap object you can draw on the surface with the following functions:

void gee_draw_pixel_bm(struct geeonx_appdata *myapp, unsigned int number, unsigned int x, unsigned int y, unsigned char r, unsigned char g, unsigned char b)

void gee_draw_box_bm(struct geeonx_appdata *myapp, unsigned int number, unsigned int x, unsigned int y, unsigned int w, unsigned int h, unsigned char r, unsigned char g, unsigned char b)

void gee_draw_line_bm(struct geeonx_appdata *myapp, unsigned int number, unsigned int fx, unsigned int fy, unsigned int lx, unsigned int ly, unsigned char r, unsigned char g, unsigned char b)

Please be aware that the surface has got is on coordinate system.

window

0,0

w,h

bitmap object

Please note also that the function gee_draw_line_bm has got a clipping feature. But gee_draw_pixel_bm and gee_draw_rectangle_bm haven't. Hence, drawing over the borders will provoke a memory violation !

> Please note also that the function gee_draw_line_bm has got a clipping feature. But gee_draw_pixel_bm and gee_draw_rectangle_bm haven't. Hence, drawing over the borders will provoke a memory violation !

- 9 -
Compiling

You can use or adapt the options out of the makefile provided with your Geeonx package:

For Linux users:

geeonx_demo:

    gcc `sdl2-config --cflags` -m64 -c geeonx_demo.c

    gcc `sdl2-config --libs` -o geeonx_demo geeonx_demo.o -lSDL2 -lSDL2_ttf -lgeeonx

For Windows 10 users:

Via command line on MSVC:

cl /c geeonx_demo.c

cl geeonx_demo.obj libgeeonx.lib SDL2.lib SDL2_ttf.lib /link /out:geeonx_demo.exe

Important: Please take care that copies of the Geeonx icons

icon_move.png, icon_close.png, icon_switch.png, icon_left.png, icon_right.png, icon_up.png, icon_down.png

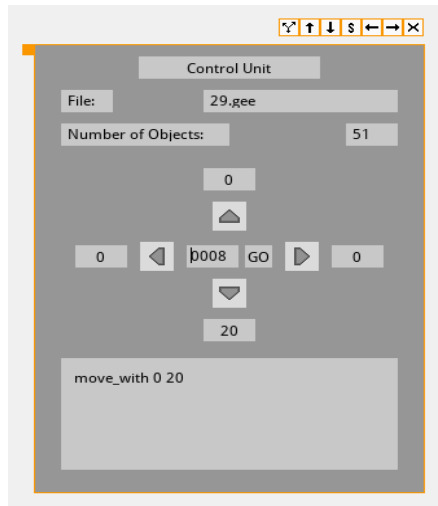and your .gee and .gew file are in the path of your application.

- 10 -

Using Geeonx Creator

After starting Geeox Creator you should open the file template.gee or template_wselector.gee, if you like to work with the Geeonx fileselector. The Geeonx objects 1 – 7 are used for the window operators. The objects 8-14 and 20 are used for the fileselector.

1. Select the object to be edited
Geeonx itself will first select the first mother_object as selected _object. You can select yourself the Geeonx object to edit with the Geeonx control unit.

The input form in the center shows the selected object. If there is a mother_object, it will be shown above the up-arrow. The first daughter object (stored in move_with[0]) is displayed under the down-arrow. Sister objects are displayed

next to the left- or right-arrows. You can move through the Geeonx objects with the arrows and/or by typing the number of the wanted object and click the Go-button afterwards.
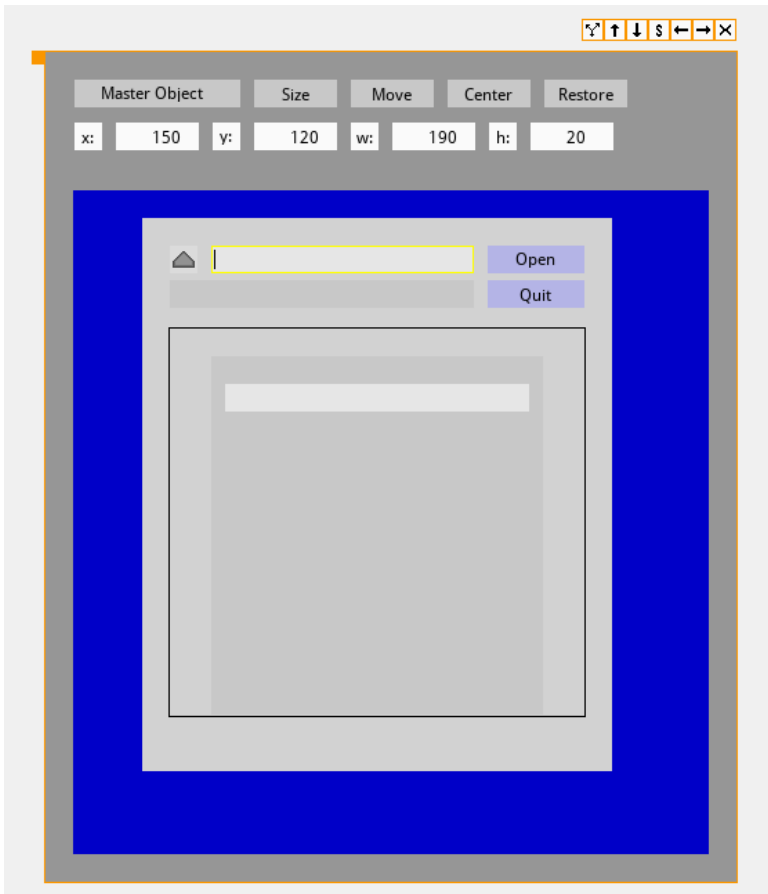

2. The show-window

The working_object will be displayed in a show-window:

In this example the Geeonx fileselector is the selected object. The x, y, w and h values of the selected object are shown in the white displays.

After clicking on the "Size" button you can easily change the size of the object by moving the mouse arrow. After clicking the w and h values are changed. With Restore you can restore the values.

The view modus which is switched on 'Selected Object' as you can see at the button in the top of the window. If you click on the button, Geeonx Creator will switch into 'Master Object' view mode. In this case the master_object of the edited object is shown. A master_object is the mother_object in a chain of of mother_objects that itself hasn't got any mother. This is normally a window. The selected_object itself is outlined yellow. Here it is the editing field:
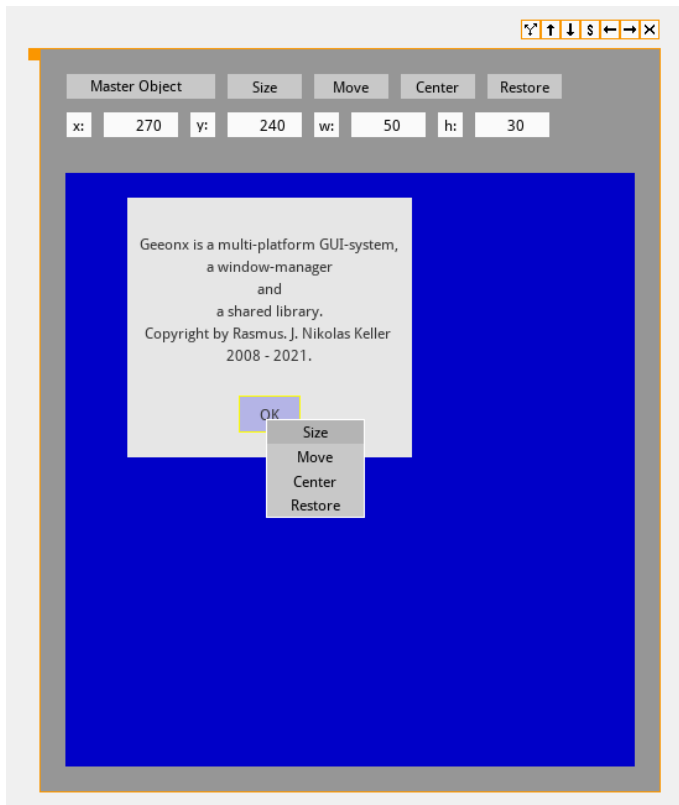
If the selected_object is already a master_object, it is not possible to switch into master view mode.

3. Editing objects in 'Master Object' mode

The size of an object can be changed visually via mouse movement by clicking the 'Size'-Button in the show-window. If the show-window is switched to the 'Master-Object' mode, the selected depending daughter object can also be moved with the mouse after selecting the 'Move'-button. With 'Restore' you can reset the position to the old position.

By right clicking on the the selected object you can choose the function you want to use from a pop-up menu:
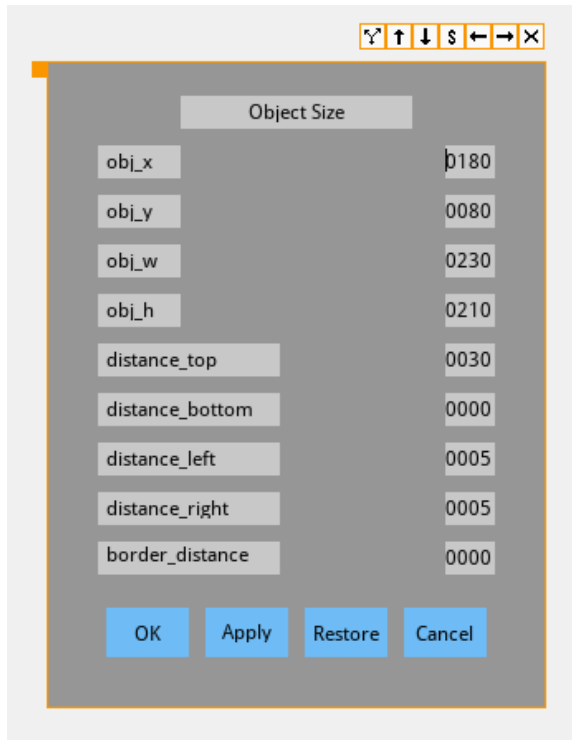
4. Editing objects with dialogs

You can edit some basic values within the Object Status dialog of the Object Menu. Furthermore you should edit the size of the object and the text space within the Object Size menu. You can edit text parameter within the dialog with the same name. All the editing dialogs are placed in the Object pull-down-menu.

You can apply the values of the form with 'Apply'. By choosing 'OK' you apply the values and close the form. With 'Restore' you can get back the old values. To reuse the old values you have to 'Apply' again.

5. Erase, copy objects

In the Action pull-down-menu you find the dialogs for the above mentioned actions.

6. Add or take away objects

With the functions add or take away objects in the Action menu you can increase or decrease the number of objects to edit.

Rasmus J. N. Keller 31.07.2022