

# **programmers**

Thomas Landspurg

<b>COLLABORATORS</b>
----------------------

	<i>TITLE :</i> programmers	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY	Thomas Landspurg	March 3, 2023
<i>SIGNATURE</i>		

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>programmers</b>	<b>1</b>
1.1	SuperDark prog . . . . .	1
1.2	Overview . . . . .	1
1.3	Info . . . . .	2
1.4	Dark function . . . . .	2
1.5	proc_init() . . . . .	3
1.6	MyGadg . . . . .	3
1.7	end_liste . . . . .	5
1.8	button . . . . .	5
1.9	slider . . . . .	5
1.10	checkbox . . . . .	5
1.11	mx . . . . .	5
1.12	cycle . . . . .	6
1.13	listview . . . . .	6
1.14	own_gadget . . . . .	6
1.15	string . . . . .	6
1.16	integer . . . . .	6
1.17	screen . . . . .	7
1.18	font . . . . .	7
1.19	data_string . . . . .	7
1.20	image . . . . .	8
1.21	exemple . . . . .	8
1.22	Info_text . . . . .	8
1.23	Problems . . . . .	9

# Chapter 1

## programmers

### 1.1 SuperDark prog

Writing your own modules for superdark:

Overview

Info

Dark function

MyGadg

The info text

Problems

### 1.2 Overview

I Overview:

How to add modules to SuperDark.

- It is not very difficult to add your own module to Superdark, but you have to read carefully this doc, because there is some important things to understand!

You have to do four things to add a new module in SuperDark:

\* Create your own

```
dark()
```

procedure, that's the procedure called at screen blanking. That's the biggest part of the job.

\* Add the

```
proc_init()
```

```
,
```

---

```

        proc_save()
        ,
        proc_end()
        function. Usually, these
function are empty. But for special purpose, you could fill them, look
at the doc.
* Add a
        my_gadg[]
        array. This array is used to create the param window,
but also to read and save parameters. Bery powerful, and easy to use
after ypu've read the doc!
* Create an
        info
        text. This should be easy,no?

```

#### Notes:

\* YOU MUST link your module with the `proc_main.o` object. `Proc_main` will make some important initialisation, contain important function for you!

\* I use some special keyword of the Lattice compile, like `_saveds`. If you don't have them, check your compiler's documentation.

The simplest way to add a new module is to take a look at the module named "`rien.dark.c`", this module does aboslutely nothing, but you can use it at a skeletton program.

## 1.3 Info

II What is a module for SuperDark ?

- It's an executable program, loaded by SuperDark.
- You have link it with `proc_main.o`.
- `Proc_main.o` will make initialisation and wait for a signal from `superdark`, and then will call your `dark` function.

## 1.4 Dark function

III

The `dark()` function:

This function is called by `proc_main` when the screen should be blanked. You can ask for a copy of the frontmost screen. To do this, look at the

```

proc_init()
function.

```

So you can put what you want here, but remeber these limitations:

- Don't use any `printf` in your program
-

- Don't make any global auto-initialised variable. This may cause some problems if you don't see exactly what happens. Because when your dark() function is called two time, the global are not reset to her initial value. So if you use global var and change their value, be careful.
- Try to don't use the whole CPU time. Use WaitTOF(), Delay(), Wait()... But you are a good programmer, so you know what to do!

To test if your function have to exit, you have the tst\_end() function. This function will return TRUE if you have to exit, or FALSE if you have to continue.

Example:

```
while(tst_end()==FALSE){
    do what you want...
}
```

The other function is wait\_end(). This function will only return at the end of the blanking time. ←

## 1.5 proc\_init()

IV proc\_init(), proc\_save() and proc\_end()

proc\_init() is called after your module have been loaded. You can do special initialisation here, but don't use to much memory. That's here that you told if you want a copy of the current frontmost screen be opened for you. To do this, you have to do this:

```
p_data_proc->type_screen=SCR_GIVEN;
```

You can alos tell to superdark that your module should only run with superdark for WB2.0 or higher.

```
p_data_proc->code_ret=DARK_WB_20;
```

proc\_save() is no more used, but is still here for compatibility...

proc\_end() is called when the module have to be removed from memory. You can free what you've allocated in proc\_init().

## 1.6 MyGadg

V The my\_gadg array....

This one of the most powerful of superdark. Each module can have a parameter window. To do this, you have to define this window, but in a special manner. This array is an array of type tom\_gadget. This is the definition of the type tom\_gadget (from includes/tom\_gadget.h) :

```
typedef struct tom_gadget{
    char *GadgetText; /* Text of the gadget on screen. */
    type_gadget type_gadg; /* The type of the gadget, see later */
}
```

```

short int LeftEdge,TopEdge; /* position      */
short int Width,Height; /* size           */
short int value; /* value !           */
short int d1,d2,d3; /* Used for special purpose.... */
char *p_data; /* Used for special purpose.... */
struct Gadget *p_gadg; /* Internal use only */

};

```

Note that the four standart gadget "ok","test","cancel","help" are automatically added!

Let's see these field:

```
char *GadgetText:
```

So it's the text of your gadget in the configuration window, but also of the corresponding value in the TOOL TYPE list. Yes, because SuperDark se these informations to save the configuration as ToolTypes in the .info of the module.

You can use shorcut, by using '\_' . Example, a gadget named TEST can have the shortcut T if his name is:

```
"_TEST"
```

```
type_gadget type_gadg:
```

Define the type of the gadget. These type are available:  
(note that they are also availble on WB1.3 for the most of them)

The following type are available(and look at the exemple

```
:
```

```
END_LISTE
```

```
BUTTON
```

```
SLIDER
```

```
CHECKBOX
```

```
MX
```

```
CYCLE
```

```
LISTVIEW
```

```
OWN_GADGET
```

```
STRING
```

```
INTEGER
```

```
SCREEN
```

```
FONT
```

```
DATA_STRING
```

```
IMAGE
LeftEdge,TopEdge,Width,Height:
Position and size of the gadget...nothing else to say!
```

## 1.7 end\_liste

```
END_LISTE:
```

This is not really a type, but each tom\_gadget array should end with this value! Remember it!!!

## 1.8 button

```
BUTTON:
```

A simple button. You can define a function called when the button is pressed, by giving a pointer to this function in the p\_data field.

## 1.9 slider

```
SLIDER:
```

A normal slider. The direction of the slizer (horiz/vertical) is defined by the ratio Width/Height. If Width is > Height, this will be an horizontal slider, otherwise it's an vertical one.

\* value must contain the initial value of the slider,

\* d1 and d2 must contain the range of the slider

\* if p\_data is not nul, it must contain a pointer to the variable wich will receive a copy of the current value of the slider.

## 1.10 checkbox

```
CHECKBOX:
```

Checkbock gadget.

\* value contain the initial value, and will be re-actualised,

## 1.11 mx

---



**MX:**

Multiple choice gadget.

\* p\_data must contain a pointer to an array of char  
example: p\_data={"Choice1", "Choice2", "Choice3", NULL}

\* value contain the initial value, and will be re-actualised,

## 1.12 cycle

**CYCLE:**

Cycle gadget.

Same kind of configuration than the MX gadget.

## 1.13 listview

**LISTVIEW:**

Listview gadget.

\* p\_data must contain a pointer to a List structure. the name of each node of the list will be show on screen.

\* value contain the position of the initial selected value, and will be re-actualised.

## 1.14 own\_gadget

**OWN\_GADGET:**

Very special purpose gadget!!!!

No time to describe it now...sorry

## 1.15 string

**STRING:**

String gadget

\* p\_data must contain a pointer to a buffer of char. This buffer will contain the value of the string gadget.

\* dl MUST CONTAIN the size of the buffer.

## 1.16 integer

---

**INTEGER:**

Integer gadget.

\*value contain the initial value, and will be re-actualised,

\*d1 contain the max number of digits for the number.

## 1.17 screen

**SCREEN:**

High level gadget....

This gadget will only be active if reqtools.library v38 or higher is present on your system.

It allow you to choose a screen resolution, size, and overscan value, by using the screen requester from the reqtools lib.

\* value contain the initial Overscan mode of the screen, and will be re-actualised. Look at intuition/screens.h

\* d1 contain the initial Width of the screen, and will be re-actualised

\* d2 contain the initial Height of the screen, and will be re-actualised

\* d3 contain the initial Depth of the screen, and will be re-actualised

\* p\_data contain the initial Mode ID of the screen, and will be re-actualised

Note:

If d3 (depth) is null, this mean that none of the value is significative.

## 1.18 font

**FONT:**

The second high level gadget....

This gadget will only be active if reqtools.library is present on your system.

It allow you to choose a font, include size and attributes.

\* p\_data is a pointer to a TextAttr structure.

IMPORTANT NOTE: This text attr must have the field ta\_Name initialized with a pointer to a string buffer, with enough space to put the biggest name of the available font

## 1.19 data\_string

**DATA\_STRING:**

This data type is only used for configuration. I mean than you won't see anything on configuration window, but a string will be saved and loaded in the configuration file.

---

## 1.20 image

### IMAGE

With this data, you can include an image in your parameter window. Just put a pointer to an image structure in the p\_data field

## 1.21 exemple

Let's we want to make a parameter window, with three things:

- a checbox,
- a slider , wich can have value from -10 to 50
- a "screen" button.

```
struct tom_gadget my_gadg[]={
{"_Check it"      ,CHECKBOX  , 100, 10, 10,10,0,0,0,0,0},
{"_slide it"     ,SLIDER   , 100, 25, 10,10,10,-10,50,0,0},
{"s_creen"       ,SCREEN   , 200, 10, 50,10,0,0,0,0,0}};
```

The parameter window will look like this:

```
-----
|           |
|  _  _____  |
| Check it  | |      |screen|  |
|  -  - - - - -  |
|  _____  |
| slide it  |          #  |  |
|          - - - - - - -  |
|           |
|      Ok      Test      Cancel      Info |
|           |
|-----
```

and when you press save, you'll see in the tooltip of your module:

```
CHECK IT=FALSE
SLIDE IT=20
SCREEN =320 x 256 x 3 ID=21000 OVSCN=0
Nice no?
```

## 1.22 Info\_text

The information text:

It's the text show when you press the "Info" button. It's just an ascii string, named p\_text\_info. Example:

```
char *p_text_info=
" Hi evrybody\n"
"now I can do my\n"
```

```
"own module for\n"  
"    SUPERDARK!\n";
```

Note the \n at the end of each line

## 1.23 Problems

VI If it doesn't work....

Hum....take a look at the other sourcecodes in the disk

If this still doesn't work, you can send me you module and I'll try to make it work....

my adress

```
Thomas LANDSPURG  
9, Place Alexandre 1er  
78000 VERSAILLES  
FRANCE
```

```
FidoNet: 2:320/104.18  
AMyNet: 39:180/1.18  
UseNet: Thomas_Landpsurg@ramses.gna.org
```

SuperDark may not be included with any commercial product nor may it be sold for profit either separately or as part of a compilation without my permission. It may be included in non-profit disk collections such as the Fred Fish collection. It may be archived & uploaded to electronic bulletin board systems as long as all files remain together & unaltered.

---