programmers

Thomas Landspurg

COLLABORATORS			
	TITLE : programmers		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Thomas Landspurg	March 2, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

programmers

1.1 SuperDark prog

```
                Writing your own modules for superdark:


               Overview

               Info

               Dark function

               MyGadg

               Sample code

               Debug

               The info text

               More info

               Writting ASM modules

               Problems

               Important info
                For old modules!
```

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 carefuly 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. Very powerful, and easy to use
  after you've read the doc!
* Create an
                info
                 text. This should be easy,no?

* You can
                Debug
                 using SDprintf()
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.But if you
don't use callback procedure with button, you don't need this.

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

Note: You should use the function DCloseScreen() instead of CloseScreen().
This function will keep the last opening screen in random mode. So in
this mode, the workbench screen will not appears between two modules.
```

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.

But:

- You can't do ANY printf
- You can't run it alone
```

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-initalised variable. This may be 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.
  A new function is available since superdark v2.0, this function is called
SDWait() (like SuperDarkWait).

    ULONG  SDWait(ULONG sigmask);

    sigmask is mask of the signal you are waiting for. SDWait will return
if one of your signal is availbale. It's exactly the same than the exec
Wait() function, but this function also do some internal SD function
(like CPU watchdog), so IF you want to use  a wait function, YOU HAVE to
use this function.

Return code: Since SuperDark v2.0, the dark function allow a return code.
This code is zero if evrything is ok, non zero otherwhise. This will be
used by the main superdark function to give some information to the user.
So try to give a return code with your module.
```

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 too 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 also 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
automaticly added!

Let's see these field:

char  *GadgetText:
  So it's the text of your gadget in the configuration window, but also
of the corrseponding 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 availaible:
(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, otherwhise 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 containt a pointer to a List structure. the
  name of each node of the list will be be show on screen.

  * value contain the position of the initial selected value,
  and will be re-actulised.
```

1.14 own_gadget

```
OWN_GADGET:
  Very special purpose gadget!!!!
  No time to describe it now...sorry.

  I just can say that it's used to create other type of
  gadget than the current available.
```

1.15 string

```
STRING:
  String gadget

  * p_data must contain a pointer to a buffer of char. This buffer
  will containt the value of the string gadget.

  * d1 MUST CONTAINT the size of the buffer.
```

1.16 integer

```
INTEGER:
  Integer gadgt.

  *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 you have WB3.0, or 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
  * d2 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....
  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 inv_checkbox

```
INV_CHECKBOX:
  This stand for "invisible checkbox". This type of gadget
  IS NOT a gadget! This will do nothing on screen,but it's
  just used to save/load a bool value. This value can only
  be changed by user using the info menu of the workbench
  on the superdark.info.
```

1.22 exemple

```
  Example: 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 OK, you'll see in the tooltype of your module:

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

1.23 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 the great\n"
"    SUPERDARK!\n";

  Note the \n at the end of each line
```

1.24 example

Sample code:

First, take a look at the sample blankers source code. This should give you some information. Here is a simple exemple of a blanker. This blanker do absolutly nothing!:

```
/* ---  start of code ---- */

#include  <exec/types.h>
#include  <exec/ports.h>
#include  <dos/dos.h>
#include  "/includes/struct.h"
#include  "/includes/tom_gadget.h"

extern  struct  RastPort  *rp;
extern  struct  Screen    *s;
extern  struct  appel_proc  *p_data_proc;

char  *name_module="vide.dark";
char  *p_text_info=
"Rien\n"
"This programm do nothing\n"
"It's a skeletton for your own\n"
"purpose";

/* An empty list of gadget....*/
struct  tom_gadget  my_gadg[]={
  {0,   END_LISTE,  0,  0,   0, 0, 0,0,0,0,0}};

/* Put here your own code */

void  dark()
{
  wait_end();
}

/*************************************************************************/
/* These 3 function have to be here, even if they are empty     */
/*************************************************************************/
void  proc_init()
{
}
void  proc_save()
{
}
void  proc_end()
{
}

/* ---  end of code ---- */
```

To create a module, do this, using lattice (suppose your file name is my_blanker.dark):

```
lc my_blanker.dark.c
```

```
blink FROM LIB:c.o proc_main.o my_blanker.dark.o LIB $(LD_LIBS) SC SD TO  ↩
    my_blanker

Then, configure "Dark directory" in superdark to the directory
where is this blanker. So you should see it in the file list modules.
```

1.25 debug

```
  Now, a little help to debug your module....

- First, try to make your 'effect' alone, without using superdark. Then,
  put it as a blanker.

  SHOW_INFO:
  ----------
  Usage:  show_info

- There is an utility program, called show_info. This program, open two
  window. In the first window there is some not so interesting informations,
  but the second window is a text window. And you can send text to this
  window, by using a SDprintf() function, like this:

    SDprintf("Hello, I'm in my module, and the value is:%ld\n",toto);

  NOTE: DO NOT USE printf, use SDprintf

  NOTE: The function use the exec RawDoFmt, but this last function only
        work with LONG value, so use %ld,%lx, instead of %d,%x, etc...
```

1.26 more

```
some more informations:

- CPU timeout:

   Since v2.0, there is a check to see if there is enough cpu time to
run the blanker. This features will automaticly be added using the last
version of proc_main.o. The older version (module which maybe are
not recompiled) will not have this fetures, but will run correctly.
   This function work fine if you use the tst_end() or wait_end()
functions. If you don't use them (REALLY? Strange...) or IF YOUR PROGRAM
SHOULD ABSOLUTLY NOT HAVE THIS "CPU TIMEOUT" WATCHDOG, add this line
in your proc_init() function:
  p_data_proc->enable_watchdog=FALSE;

- Version:

  Since V2.0 of superdark, there is a version number available for
the module. This version can be found in p_data_proc->version (look
at file "struct.h"), and revision number is in p_data_proc->revision.
So first version available will be version 2, revision 0.
  This prevent some module to use some features not available in older
```

version of superdark.

1.27 ASM

I've just talk of C, but the module can be made in any langage.
You can write an ASM module. But you still have to link it with
proc_main.o. You can do this using the great devpac assembler.
But there is no tom_gadget.i, only tom_gadget.h. That's why
you could use the same method than me: make a c program where
there is evrything except the dark() function, and then put
the _dark() function in an asm program, and them link evrything!
Just take a look at the plasma sourcecode, in the full superdark
distribution.

1.28 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

1.29 Old_Versions

This is very important:

If you've made a module for superdark less then 2.1 (1.2,1.3..to 2.0b)
YOU HAVE TO RECOMPILE IT USING THE LAST proc_main.o. Because the

communication method have changed! If you don’t do this, your old module will not work!!!!