

# GUI Script

## User Manual

Document version 1.05



## Revision History

Version	Date	Changes
1.00	15 Nov 2010	Original version, used with GUI_ENGINE_V1_00.lib gui_engine_v3_00.h gui_sound_plugin_v2_00.h
1.01	20 Nov 2010	More description about GUI SCS W GUI CFG S added Section added about sound plug-in functions Used with: GUI_ENGINE_V1_01.lib gui_engine_v3_01.h gui_sound_plugin_v2_01.h
1.02	17 Feb 2011	New features added to screen New functions added in 10.3 New programming guidelines added in 10.6 Used with : GUI_ENGINE_V1_07.lib gui_engine_v3_07.h gui_sound_plugin_v2_03.h
1.03	26 Mar 2011	New compiler added in chapter 9 New functions added New chapter “Extension Script” added Appendix added Command list moved to appendix Used with : GUI_ENGINE_V1_09.lib gui_engine_v3_09.h gui_sound_plugin_v2_05.h
1.04	22 July 2011	Popup and percent bar added Used with : GUI_ENGINE_V2_00.lib gui_engine_v3_10.h gui_sound_plugin_v2_05.h

Version	Date	Changes
1.05	2 Dec 2011	Embedded script added Default alignment of textbox, label and table are fixed Transparency of textboxes supported New functions added New chapter “SUN7 Studio” added Used with : GUI_ENGINE_V2_05.lib gui_engine_v3_16.h gui_sound_plugin_v2_05.h

## Contents

1. Introduction.....	7
2. Related Files.....	7
3. Features .....	8
4. Commands and Parameters.....	11
4.1 Command Structure .....	11
4.2 Parameter Descriptions .....	11
5. GUI Commands .....	13
5.1 RBG: Register background images. ....	13
5.2 RFONT: Register fonts.....	13
5.3 RXFONT: Register extension fonts.....	14
5.4 RIFONT: Register image fonts.....	14
5.5 RIMG: Register images. ....	15
5.6 MEM: Show memory status. ....	15
5.7 SCS: Screen settings. ....	15
5.8 BG: Background image settings. ....	19
5.9 IMG: Image box settings. ....	20
5.10 BT: Button settings. ....	21
5.11 TXT: Textbox settings. ....	27
5.12 LBL: Label settings. ....	30
5.13 BOX: Box settings. ....	32
5.14 TAB: Table settings. ....	33
5.15 CFG: Configure global parameters.....	35
5.16 SND: Play a MP3 file. ....	36
5.17 END: End all settings. ....	36
5.18 BEG: Set startup options.....	36

5.19 BAR: Percent bar settings.....	37
6. SND Commands .....	43
6.1 CFG: Configure global parameters.....	43
6.2 RG0: Register MP3 files.....	43
6.3 RGC: Register MP3 files for multi-language application. ....	44
7. Writing Script.....	45
7.1 Script Structure .....	45
7.2 Script Making Guidelines. ....	47
8. GUI Engine Setup Guide .....	48
8.1 Download Example Project .....	48
8.2 Add GUI Engine Library .....	48
8.3 Disabling Sound Plug-in.....	50
9. Application Programming Interface .....	51
9.1 Setting Parameters .....	51
9.2 Object Structures.....	54
9.3 API Functions (GUI) .....	63
9.4 API Functions (Sound) .....	78
9.5 External Variables.....	79
9.6 Programming Guidelines .....	80
10. Extension Script .....	84
10.1 Commands in Extension Script .....	84
10.2 Attach Script to Components.....	84
10.3 Events.....	85
11. SUN7 Studio .....	86
11.1 To get SUN7 Studio.....	86
11.2 User Interface Description .....	87
11.3 To create new project.....	105

Appendix .....	106
1. Installation of Eclipse and Yagarto.....	106
2. Command List.....	110

## 1. Introduction

GUI script is a new concept for GUI development on non-OS application. It's designed for engineers who have skill in C programming and microcontrollers. GUI script makes GUI development faster, upgradable and more flexible. Advantages of using GUI script are described in **BlueScreen SUN7 User Manual**. In this user manual user will have more detail on commands, API and programming guidelines. The last chapter introduces and describes how to use **SUN7 Studio**. The latest software tool developed to create GUI graphically.

Since version 1.09, GUI Script supports “**Extension Script**” for very simple projects which basic control is performed, for example, driving some output ports or simple communication through serial port.

## 2. Related Files

Source files related with GUI script are:

[gui\\_engine.h](#): The header of GUI engine, it processes GUI commands from script file.

[gui\\_sound\\_plugin.h](#): In application with sound needed, this is a plug-in that works cooperatively with [vs1011.c](#) to manage MP3 files playing along the screens.

[gui\\_script\\_bridge.c](#): With library limitation, parameters defined must be stored somewhere in memory so library can be used with various settings. This file stores defined settings in code memory and will be read by the GUI engine.

[app\\_scr\\_func.c](#) (and .h): This is the application specific source file created for users to writing their code. And users need to define quantity of objects in the header file.

[GUI\\_ENGINE.lib](#): The library file.

The source file versions this user manual referenced to are issued in Revision History. Changes in source files can be seen in their header files.

### 3. Features

#### Screens:

- Wait time assignable for no activity screens
- Namable (assign name to screens for easy identification)
- MP3 files assignable to play at the beginning of screens
- Multiple auxiliary values assignable for user purpose
- Standard keypads selectable to screens
- Image backgrounds with language assignable or plain color backgrounds
- Event before screens being drawn
- Event after screens being drawn
- Event every 100ms while the screen being displayed
- Event when keypad pressed
- Event before key textbox inserted
- Event when sound list ended
- Extension script supported

#### Popup screen:

- Same features as normal screens
- Unlike normal screen, popup screen is just a window (not full screen) shown on a normal screen and when closed, the base screen continues as it was before the opening of the popup screen

#### Buttons:

- Image buttons with language assignable
- Plain color buttons
- Image or color in normal, pressed and disabled states
- Transparency supported
- Movement (while being pressed) supported in downward and rightward direction only
- Namable (assign name to buttons for easy identification)
- On-button text with specifiable font, color and bold characteristic
- Key buttons supported
- Language mode change supported
- Auxiliary value assignable for key buttons or user purpose
- Go to target screen ID when released or open/close popup screen
- Shift interlock buttons
- Global sounds played when buttons are pressed



- Event when pressed
- Event when released
- Event when buttons are associated in all case
- Extension script supported
- Repetition supported: events repeated while pressing

#### Images on screen:

- Language assignable
- Transparency supported
- Namable (assign name to image boxes for easy identification)

#### Textboxes:

- Text, background, and border color assignable
- Maximum characters assignable
- Password mode with assignable displayed character
- Specifiable font, color and bold characteristic for texts
- Text alignment: left, center and right
- Insertion allowed (with cursor enabled) or not (no cursor)
- Image font supported
- Namable (assign name to textboxes for easy identification)
- Transparency supported

#### Tables:

- Row quantity, row height assignable
- Column width, caption color, caption text and content color assignable individually
- Specifiable font, color and bold characteristic for texts
- Border color assignable
- Event when pressed
- Event when tables are associated in all case
- Namable (assign name to tables for easy identification)

#### Labels:

- Language assignable
- Alignment: left, center and right referenced from origin position
- Specifiable font, color and bold characteristic for texts
- Namable (assign name to labels for easy identification)
- Image font supported

#### Percent Bars:

- Vertical and horizontal bar supported
- Bar with pin supported
- 4 touch response styles
- 2 movement styles
- Namable (assign name to bars for easy identification)
- Event on the move
- Event when moving stopped

#### Boxes (for making simple boxes or lines on screen):

- Color assignable

#### Fonts & Extension fonts:

- Accepts binary font files generated from bmp2h\_conv software

#### Language:

- Supports up to 8 language modes

#### Paths:

- Support multi-path applications, if the hierarchy can be divided into paths and sharing some common screens, path can be set for easier implementation

## 4. Commands and Parameters

### 4.1 Command Structure

GUI script is processed by console; the command used here is “GUI” (just like “LS” command that will show content in current directory). But with so many components to be set for GUI, a subcommand is needed. Each subcommand needs different parameters. Below is the command structure:

```
GUI subcommand parameter1 parameter2 ... [parameterX]
```

Please note that:

1. Only “GUI” is not case-sensitive, since it is just a console command. **The rest is case-sensitive.**
2. With console version 2.04 or later, **you can use *space* or *tab* to separate tokens.**
3. With console version 2.04 or later, **you can assign a number in 3 formats: straight number (e.g. 123, 4) character format (e.g. ‘a’, ‘A’) or hexagonal format (e.g. 0xA0, 0xFFFF).**
4. Parameters written in [ ], are optional, if there is in a command, it’s always the last one. Unless they are set, default value will be used.
5. Every parameter that is text type (file paths, names, texts), may contains spaces. Console engine accepts all characters until the end of line found. This kind of parameters is always the last one in commands.

### 4.2 Parameter Descriptions

- Color: color value range depends on color depth used in the application. Color depth can be 8, 16 or 24 bpp. Below are example color values (also listed in [lcd\\_ctrl.h](#)). Note that default color is black (0), parameters don’t need to be set if default value used.

*Common color table*

Color	8 bpp	16 bpp	24 bpp
Red	0x03	0x001F	0x0000FF
Green	0x1C	0x07E0	0x00FF00
Blue	0xE0	0xF800	0xFF0000
Yellow	0x1F	0x07FF	0x00FFFF
Purple	0xE3	0xF81F	0xFF00FF
Black	0x00	0x0000	0x000000
White	0xFF	0xFFFF	0xFFFFFFFF

- Language: language variable in software is char-type. It's always one in 1,2,4,8,16,32,64,128 ( $1 \ll n$ ,  $n=0-7$ ). Anyway, the language value can be set from 1 to 255. This allows an object supports more than 1 language. In general case, use 1 2 8 ... for single language objects and 255 for objects used for all language.

- Object ID: screen ID, button ID and so on, are indexed by short-type variable. A short-type variable consumes 2 bytes, so it can be 0-65535 (0xFFFF is 65535). Anyway, the value 65535 is used as some unset value and shouldn't be used by users.

## 5. GUI Commands

### 5.1 RBG: Register background images.

The concept of GUI script is simple, images, background images and fonts must be first read and stored to SDRAM. An ID number must be assigned to each of them. And after that, this ID number will be used in the rest of script.

```

GUI RBG ID Image_type Lang Path
e.g.      GUI RBG 0  2  255  img/bgl.bmp

ID:        ID number of background image
Image_type: type of image;
            0 for binary files (.bin)
            1 for JPEG files
            2 for BMP files
Lang:      language for background image
Path:      path of the file stated from current directory containing
            script file, for example, if the script file path is
            "sr/main.txt", here the image path must be "sr/img/bgl.bmp"
  
```

### 5.2 RFONT: Register fonts.

```

GUI RFONT ID Width Height Gap Path
e.g.      GUI RFONT 0  25 24 1 fonts/arial_eng25x24.bin

ID:        ID number of font
Width:     width of character block in pixel
Height:    height of character block in pixel, can only be 8, 16, 24..
128
Gap:       Gap between characters for display
Path:      see RBG command
  
```

### 5.3 RXFONT: Register extension fonts.

```
GUI RXFONT ID Width Height Adj Aux Path
```

e.g. GUI RXFONT 0 25 24 1 0 fonts/arial\_thai25x24.bin

ID: ID number of extension font  
 Width: width of character blocks in pixel  
 Height: height of character blocks in pixel; can only be 8, 16, 24...  
 128, anyway, Thai fonts are supported up to 64.  
 Adj: height adjustment for English font and extension font  
 Aux: auxiliary value, use dependently on extension language (for  
 Thai, this is the height for shifting tone when upper vowel  
 is presented)  
 Path: see RBG command

### 5.4 RIFONT: Register image fonts.

```
GUI RIFONT ID 1st_char nChar Image_ID Gap
```

e.g. GUI RIFONT 0 '0' 10 5 1

ID: ID number of image font  
 1<sup>st</sup>\_char: first character of font  
 nChar: number of character, for example a numeric font have 10  
 characters (0-9)  
 Image\_ID: ID of image registered with "RIMG", only the first one  
 needed to be assigned (images must be registered  
 respectively)  
 Gap: Gap between characters for display

## 5.5 RIMG: Register images.

```

GUI RIMG ID Image_type(0) Width Height Lang Path
GUI RIMG ID Image_type(2) Lang Path
GUI RIMG ID Image_type(3) Width Height Lang Color [Border]

e.g.      GUI RIMG 0 0 25  75 255 img/button0.bin
          GUI RIMG 1 2 255 img/button0.bmp
          GUI RIMG 2 3 25  75 255 0xFFFF

ID:        ID number of image
Image_type: type of image;
            0 for binary files (.bin)
            1 for JPEG files, currently not supported
            2 for BMP files
            3 for plain color button
Width:     width of image in pixel
Height:    height of image in pixel
Lang:      language for image
Color:     color of button body
Border:    color of button border, only applied when
           BT3_SMOOTH_LV = 0 (see obj\_lib.c, default is 2)
Path:      see RBG command
  
```

## 5.6 MEM: Show memory status.

This should be places after all file registering finished, where all memory used for images, fonts and extension fonts are summarized.

```
GUI MEM
```

## 5.7 SCS: Screen settings.

### - SCS S

Set associated screen ID, with this command, scripts afterward will belong to this screen ID.

```

GUI SCS S ID

e.g.      GUI SCS S 0

ID:       screen ID
  
```

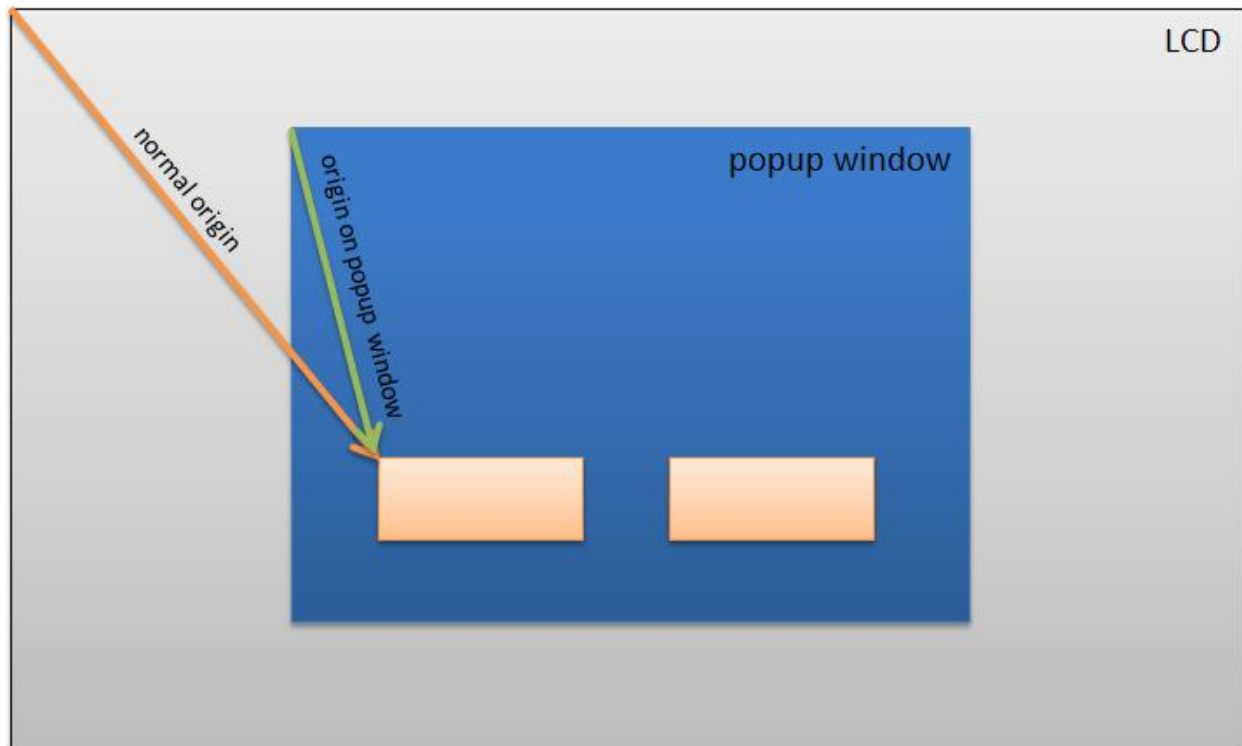
In case that the screen is popup window (not full screen), width and height of window must be set using the same subcommand.

```
GUI SCS S ID Width Height
```

e.g.      GUI SCS S 8 300 200

ID:        screen ID  
Width:    screen width  
Height:   screen height

Note that popup window always show at the center of screen calculated from its size. And all the origin positions of object inside are referenced to the window's origin (top-left), not the LCD (see image below).





### - SCS n

Name the associated screen, for easy identifying in software; user can specify the screen with a name so the screen ID can be changed without code modification.

```
GUI SCS n Name
```

e.g.        GUI SCS n screen1

Name: a text to be used as the name of screen

### - SCS P

Switch path to new path when the screen is shown up. Ignore this command for single path application.

```
GUI SCS P Path
```

e.g.        GUI SCS P 1

Path:        new path to be switched to

### - SCS C

Set char-type auxiliary values. Its range is 0-255. The quantity of value can be set in source code (default is 2), see appendix for more detail.

```
GUI SCS C Index Value
```

e.g.        GUI SCS C 0 5

Index:       index of auxiliary values

Value:       number between 0-255

### - SCS L

Set long-type auxiliary values. Its range is 0x0-0xFFFFFFFF. The quantity of value can be set in source code (default is 1), see appendix for more detail.

```
GUI SCS L Index Value
```

e.g.        GUI SCS L 0 100000

Index:       index of auxiliary values

Value:       number between 0-0xFFFFFFFF

## - SCS W

Set waiting time, in case that the associated screen may show up temporary and then switch to other screen (or switch to next screen). The count will be reset every time an object is pressed.

```
GUI SCS W Time [ID]
```

e.g.        GUI SCS W 50 10

Time:        time in 100ms, with example above, the waiting time is 5 seconds

ID:        target screen ID to be shown after the waiting completed, if ID is not entered, next adjacent screen will be shown

For multi-path applications:

```
GUI SCS W Time ID#1 ID#2 ...
```

e.g.        GUI SCS W 50 1 2 3

ID#n:        target screen ID for path 0, path 1 and so on, single path assignment means that target screen IDs are the same for all paths

## - SCS c

Set waiting time to close incase of popup screen

```
GUI SCS c Time
```

e.g.        GUI SCS c 50

Time:        time in 100ms, with example above, the waiting time is 5 seconds

## - SCS K

Enable keypad on the screen.

```
GUI SCS K Keypad_type
```

e.g.        GUI SCS K 1

Keypad\_type: Keypad types are available in 1-7  
               1 for standard 7"-screen keypad  
               2 for standard 4.3"-screen keypad  
               3-5 for future standard keypad (unavailable now)  
               6-7 for user keypad (use kp3 from [obj\\_lib.c](#))

## - SCS s

Extension script setting; a script file (text file) can be set so it is read when the screen is showed. See section 10 for more information. Allowable screen events are <INIT>, <INIT2> and <SND\_END>.

```
GUI SCS s Filename

e.g.      GUI SCS s screen1.txt

Filename:  the script file name refer to the script path, from the
example, the screen1.txt should be in "sr" folder (same location with
main.txt)
```

## 5.8 BG: Background image settings.

### - BG A

Add background images to associated screen, can be more than one (depend on number of language in the application).

```
GUI BG A ID#1 [ID#2] [ID#3] ... [ID#8]

e.g.      GUI BG A 0  1  2

ID:       ID of registered background images, the quantity of ID can
          be up to MAX_LANGUAGE defined in the code
```

### - BG C

Set background color in case of using plain color background, only one of BG A or BG C should be applied to a screen.

```
GUI BG C Color

e.g.      GUI BG C 0xFFFF

Color:    color of background, it's white in the example above (for
          16 bpp)
```

## 5.9 IMG: Image box settings.

### - IMG O

Set origin position of the image box, default is (0, 0).

```
GUI IMG O X Y
```

e.g.        GUI IMG O 100 200

X:         horizontal position  
 Y:         vertical position

### - IMG I

Set image ID used for the image box, default is 0.

```
GUI IMG I ID
```

e.g.        GUI IMG I 5

ID:         ID of registered images

### - IMG t

Set transparency, without this command, the image box will be displayed normally (showing white pixel).

```
GUI IMG t
```

### - IMG d

Disable the image at startup (can be enabled later by software).

```
GUI IMG d
```

### - IMG n

Name the image, for easy identifying in software; user can specify the image with a name.

```
GUI IMG n Name
```

e.g.        GUI IMG n image1

Name:       a text to be used as the name of image

## - IMG E

End current image box setting, end every image box setting with this command so the index increase for next one.

```
GUI IMG E
```

## 5.10 BT: Button settings.

### - BT n

Name the button, for easy identifying in software; user can specify the button with a name.

```
GUI BT n Name
```

e.g. GUI BT n Next

Name: a text to be used as the name of button

### - BT O

Set origin position of the button, default is (0, 0).

```
GUI BT O X Y
```

e.g. GUI BT O 100 200

X: horizontal position

Y: vertical position

### - BT N

Set image ID used for the button in normal state, default is 0.

```
GUI BT N ID
```

e.g. GUI BT N 0

ID: ID of registered images

### - BT P

Set image ID used for the button in pressed state, default is 0.

```
GUI BT P ID
```

e.g. GUI BT P 1

ID: ID of registered images

## - BT D

Set image ID used for the button in disabled state, default is 0.

```
GUI BT D ID

e.g.      GUI BT D 2

ID:       ID of registered images
```

## - BT I

Enable shift interlock buttons. Place 2 or more buttons on the same location, disable them all but one. The enabled one is shown and when it's pressed and released, the next one will be in place.

```
GUI BT I Order

e.g.      GUI BT O 100 200
          GUI BT S
          GUI BT N 0
          GUI BT P 1
          GUI BT I 0
          GUI BT E

          GUI BT L
          GUI BT N 2
          GUI BT P 3
          GUI BT I 1
          GUI BT d
          GUI BT E

Order:     order of the sequence, must start with 0 and can be up to 7
           (8 buttons in the sequence)
```

## - BT M

Set the movement when the button is pressed, default is 0, 0.

```
GUI BT M X Y

e.g.      GUI BT M 2 2

X:        rightward move (in pixel)
Y:        downward move (in pixel)
```

## - BT G

Set the screen to go to when the button is released. Or go to next screen.

```
GUI BT G [ID]
```

e.g.            GUI BT G 4

ID:            screen ID to go to, if not entered, go to next screen

For multi-path applications:

```
GUI BT G ID#1 ID#2 ...
```

e.g.            GUI BT G 1 2 3

ID#n:           target screen ID for path 0, path 1 and so on, single path  
 assignment means that target screen IDs are the same for  
 all paths

## - BT d

Disable the button at startup (can be enabled later by software).

```
GUI BT d
```

## - BT t

Set transparency, without this command, the button will be displayed normally (showing white pixel).

```
GUI BT t
```

## - BT i

Set inverse mode, when pressed or released button will be inversed. With this mode set, movement setting is discarded.

```
GUI BT i
```

## - BT T

Set on-button text.

```
GUI BT T Text
```

e.g.            GUI BT T Enter

Text:           text to be displayed on the button

### - BT F

Set font and/or extension font for on-button text, defaults are 0.

```
GUI BT F ID#1 [ID#2]
e.g.      GUI BT F 1 1
ID#1:     font ID
ID#2:     extension font ID (optional)
```

### - BT B

Make on-button text bold.

```
GUI BT B
```

### - BT C

Set on-button text color, default is black.

```
GUI BT C Color
e.g.      GUI BT C 0xF800
Color:     color for on-button text, blue for the example above
```

### - BT A

Set on-button text alignment, default is center (1).

```
GUI BT A Alignment_type
e.g.      GUI BT A 2
Alignment type: 0 for left, 1 for center and 2 for right
```

### - BT S

Save current button settings.

```
GUI BT S
e.g.      GUI BT M 2 2
          GUI BT S
          GUI BT N 0
          GUI BT P 1
          GUI BT O 100 200
          GUI BT E
```



## - BT L

Load settings to current button, example below shows the movement setting is copied from example above. The save and load commands help users in setting common parameters for buttons.

```

GUI BT L
e.g.    GUI BT L
        GUI BT N 2
        GUI BT P 3
        GUI BT O 300 200
        GUI BT E
  
```

## - BT V

Set value for the button. A 32-bit integer is used to store the value so the range is 0-0xFFFFFFFF.

```

GUI BT V Value
e.g.    GUI BT V 'A'
Value:  value for the button
  
```

## - BT X

Set special action for the button, default is 0.

```

GUI BT X Value
e.g.    GUI BT X 1
Value:  special action for the button:
        1 for key button, the value set by BT V will be used as a
        key character
        2 for language change button, the value set by BT V will be
        used as new language
  
```

### - BT s

Extension script setting; a script file (text file) can be set so it is read when an action occurs with the button. See section 10 for more information. Allowable button events are <PRESS> and <RELEASE>.

```
GUI BT s Filename

e.g.      GUI BT s button1.txt

Filename:  the script file name refer to the script path, from the
            example, the button1.txt should be in "sr" folder (same
            location with main.txt)
```

### - BT r

Enable repetition for the button. Repeat periods can be set with "CFG r b".

```
GUI BT r
```

### - BT p

Open popup window.

```
GUI BT p ID

e.g.      GUI BT p 8

ID:       screen ID, must be a popup screen
```

### - BT c

Close popup window.

```
GUI BT c
```

### - BT E

End current button setting.

```
GUI BT E
```

## 5.11 TXT: Textbox settings.

### - TXT O

Set origin position of the textbox, default is (0, 0).

```
GUI TXT O X Y
```

e.g.        GUI TXT O 100 200

X:         horizontal position  
 Y:         vertical position

### - TXT S

Set textbox size.

```
GUI TXT S Width Height
```

e.g.        GUI TXT S 150 50

Width:     width of the textbox  
 Height:    height of the textbox

### - TXT I

Set insertion of the textbox, disabled by default, with this command, the textbox can be pressed and the cursor is displayed. Then new characters can be inserted between the texts. Without this command (default), no cursor shown and new characters will always be appended to the right side of the texts.

```
GUI TXT I
```

### - TXT K

Set textbox active so it will receive characters from key buttons.

```
GUI TXT K
```

### - TXT B

Make text on the textbox bold.

```
GUI TXT B
```

## - TXT C

Set text and background colors, defaults are black.

```
GUI TXT C Text_color Back_color
e.g.      GUI TXT C 0 0xFFFF
Text_color: text color
Back_color: background color
```

## - TXT F

Set font and/or extension font, defaults are 0.

```
GUI TXT F ID#1 [ID#2]
e.g.      GUI TXT F 1 1
ID#1:     font ID
ID#2:     extension font ID (optional)
```

## - TXT i

Use image font. With this setting, the ID#1 from “TXT F” will refer to image font; note that bold, insertion and color settings will be ignored since character images will be displayed.

```
GUI TXT i
```

## - TXT A

Set on-text box text alignment, default is left (0).

```
GUI TXT A Alignment_type
e.g.      GUI TXT A 2
Alignment type: 0 for left, 1 for center and 2 for right
```

## - TXT L

Set maximum characters in the textbox (length).

```
GUI TXT L Length
e.g.      GUI TXT L 10
Length:    maximum of characters
```

### - TXT p

Enable password mode.

```
GUI TXT p Password_character
```

e.g.        GUI TXT p 'X'

Password\_character: a character to be shown in password box

### - TXT m

Use multi-line mode. With this mode set, the total text lines can be greater than lines the box can display. But the text length is still limited by “TXT L”. Up and down button should be created along and call **ObjMLTextBoxUp** and **ObjMLTextBoxDown** when a button is pressed or released respectively. Note that without this mode set, text can be shown only within the box. And password mode will be ignored when this mode set.

```
GUI TXT m
```

### - TXT n

Name the textbox, for easy identifying in software; user can specify the textbox with a name.

```
GUI TXT n Name
```

e.g.        GUI TXT n txt1

Name: a text to be used as the name of textbox

### - TXT t

Set transparency of the textbox.

```
GUI TXT t
```

### - TXT E

End current text box setting.

```
GUI TXT E
```

## 5.12 LBL: Label settings.

### - LBL O

Set origin position of the label, default is (0, 0). Unlike all other objects, the alignment of label refers to its origin position. So the top-left corner of the label is not its origin position when the alignment set is center or right.

```
GUI LBL O X Y
e.g.      GUI LBL O 100 200
X:        horizontal position
Y:        vertical position
```

### - LBL B

Make text on the label bold.

```
GUI LBL B
```

### - LBL C

Set text color, default is black.

```
GUI LBL C Text_color
e.g.      GUI LBL C 0
Text_color: text color
```

### - LBL T

Set label text.

```
GUI LBL T Text
e.g.      GUI LBL T Name:
Text:     label text
```

### - LBL F

Set font and/or extension font, defaults are 0.

```
GUI LBL F ID#1 [ID#2]
e.g.      GUI LBL F 1 1
ID#1:     font ID
ID#2:     extension font ID (optional)
```

### - LBL i

Use image font. With this setting, the ID#1 from “LBL F” will refer to image font; note that bold and color settings will be ignored since character images will be displayed.

```
GUI LBL i
```

### - LBL A

Set on-text box text alignment, default is left (0).

```
GUI LBL A Alignment_type
```

e.g. GUI LBL A 2

Alignment type: 0 for left, 1 for center and 2 for right

### - LBL L

Set language for label.

```
GUI LBL L Lang
```

e.g. GUI LBL L 1

Lang: language for label

### - LBL n

Name the label, for easy identifying in software; user can specify the label with a name.

```
GUI LBL n Name
```

e.g. GUI LBL n topic

Name: a text to be used as the name of label

### - LBL E

End current label setting.

```
GUI LBL E
```

## 5.13 BOX: Box settings.

### - BOX O

Set origin position of the box, default is (0, 0).

```
GUI BOX O X Y
```

e.g.        GUI BOX O 0 100

X:         horizontal position  
 Y:         vertical position

### - BOX S

Set box size.

```
GUI BOX S Width Height
```

e.g.        GUI BOX S 800 2

Width:     width of the box  
 Height:    height of the box

### - BOX C

Set box color, default is black.

```
GUI BOX C Color
```

e.g.        GUI BOX C 0

Color:     box color

### - BOX E

End current text box setting.

```
GUI BOX E
```



## 5.14 TAB: Table settings.

### - TAB O

Set origin position of the table, default is (0, 0).

```
GUI TAB O X Y
```

e.g.        GUI TAB O 50 200

X:           horizontal position  
 Y:           vertical position

### - TAB B

Make texts on the table bold.

```
GUI TAB B
```

### - TAB C

Set text and border colors, defaults are black.

```
GUI TAB C Text_color Border_color
```

e.g.        GUI TAB C 0xF800 0

Text\_color: text color  
 Border\_color: border color

### - TAB c

Set column parameters.

```
GUI TAB c Column Width Cap_color Con_color [Text]
```

e.g.        GUI TAB c 0 100 0x07E0 0xFFFF No.  
             GUI TAB c 1 200 0x07E0 0xFFFF Item Name  
             GUI TAB c 2 400 0x07E0 0xFFFF Description

Column:     column index  
 Width:      column width  
 Cap\_color:  caption color (the top row)  
 Con\_color:  content color (every row but the top one)  
 Text:       caption text (optional)

## - TAB R

Set row parameters.

```
GUI TAB R Height nRow

e.g.      GUI TAB R 50 10

Height:   row height
nRow:     number of row
```

## - TAB F

Set font and/or extension font, defaults are 0.

```
GUI TAB F ID#1 [ID#2]

e.g.      GUI TAB F 1 1

ID#1:     font ID
ID#2:     extension font ID
```

## - TAB A

Set on-text box text alignment, default is left (0).

```
GUI TAB A Alignment_type

e.g.      GUI TAB A 2

Alignment type: 0 for left, 1 for center and 2 for right
```

## - TAB n

Name the table, for easy identifying in software; user can specify the table with a name.

```
GUI TAB n Name

e.g.      GUI TAB n total

Name:     a text to be used as the name of table
```

## - TAB E

End current table setting.

```
GUI TAB E
```

## 5.15 CFG: Configure global parameters.

### - CFG L

Configure default language, default is 1.

```
GUI CFG L Lang
e.g.      GUI CFG L 2
Lang:     default (initial) language
```

### - CFG S

Configure default button sound. This will be applied for all buttons.

```
GUI CFG S ID#1 [ID#2]
e.g.      GUI CFG S 0 1
ID#1:     sound ID used when buttons are pressed
ID#2:     sound ID used when disabled buttons are pressed (optional)
```

### - CFG r k

Configure repeat interval for keypad. There are 2 intervals; the interval for first time (the interval starts when the keypad is pressed until the first action occurs) and the interval for the rest.

```
GUI CFG r k First_interval Next_interval
e.g.      GUI CFG r k 5 0
First_interval: interval between the press and the first repeat, must
                be more than 0
Next_interval:  interval between the next repeat, can be 0 for best
                effort
```

## - CFG r b

Configure repeat interval for buttons with repeat-action enabled. There are 2 intervals; the interval for first time (the interval starts when the button is pressed until the first action occurs) and the interval for the rest. At any repeats, both .press and .release events occur.

```
GUI CFG r b First_interval Next_interval
e.g.      GUI CFG r b 5 3

First_interval: interval between the press and the first repeat, must
                be more than 0
Next_interval: interval between the next repeat, can be 0 for best
                effort
```

## 5.16 SND: Play a MP3 file.

Play a MP3 file at the beginning of screen ([gui\\_sound\\_plugin.c](#) needed).

```
GUI SND ID
e.g.      GUI SND 0
ID:       registered sound ID
```

## 5.17 END: End all settings.

This command shows summarized objects created or registered.

```
GUI END
```

## 5.18 BEG: Set startup options.

### - BEG S

Set startup screen, default is 0.

```
GUI BEG S ID
e.g.      GUI BEG S 1
ID:       screen ID
```

## - BEG P

Set startup path, default is 0.

```
GUI BEG P Path

e.g.      GUI BEG P 2

Path:     path (route in the GUI flow)
```

## - BEG B

Show logo with progress bar while loading script.

```
GUI BEG B ID#1 ID#2 X Y

e.g.      GUI RBG  0 2 255 bg/bg1.bmp
          GUI RIMG 0 2 255 images/boot_bar.bmp
          GUI BEG B 0 0 25 440

ID#1:     background ID
ID#2:     image ID for progress bar
X:        horizontal position of progress bar
Y:        vertical position of progress bar
```

After this line of script executed, the background will be shown with the progress bar. It's necessary that both background and image for progress bar are registered first. And whether the `SHOW_ACTIVITY` is defined to 0 or 1 or undefined (if it's 1 the screen turns black and shows files being loaded), the screen turns to selected background instead and the `SHOW_ACTIVITY` value is ignored.

### 5.19 BAR: Percent bar settings.

Percent bars can be used to display progress status, volume and scroll bar. There are many options available, for example, the user can set its moving style, touch response, orientation. If bar is composed of tube and the bar itself, use an image (IMG) for the tube.

### - BAR O

Set origin position of the bar, default is (0, 0), always use the top-left corner no matter what the orientation is.

```
GUI BAR O X Y
```

e.g.        GUI BAR O 50 200

X:           horizontal position  
 Y:           vertical position

### - BAR I

Set image ID used for the bar, default is 0.

```
GUI BAR I ID
```

e.g.        GUI BAR I 20

ID:           ID of registered image

### - BAR v

Set orientation to vertical, default is horizontal.

```
GUI BAR v
```

### - BAR V

Set initial value (in percent), default is 0.

```
GUI BAR V Value
```

e.g.        GUI BAR V 20

Value:       initial percentage value, must be between 0-100

## - BAR s

Set touch response style, default is 0.

```
GUI BAR s Response_style [Sensitivity]
```

e.g.        GUI BAR s 2

Response\_style: numerical value, must be between 0-3  
                   0: no response (value can be set by software only)  
                   1: move to the touch (value changed only on the press)  
                   2: follow the touch movement (value changed continuously  
                     to the touch point)  
                   3: slide with touch movement (value changed continuously  
                     in the same direction with sliding)  
 Sensitivity: slide sensitivity, only applied when Response\_style is  
                   set to 3, must be between 1 and 10, default value is 3

## - BAR m

Bar movement, 0 is default value, the bar image move while its value is changing. Otherwise, setting to 1 and the bar image don't move. See images below (the bar's tube is an image that have to be shown separately).

Tube image



Bar image



m = 0, value = 50



m = 0, value = 100



m = 1, value = 50



m = 1, value = 100



```
GUI BAR m Movement_type
```

e.g. GUI BAR m 1

Movement\_type: numerical value, must be 0 or 1 (default value = 0)



## - BAR p

Pin size (in pixels). This is generally be used with movement value 0 (so pin is the head of bar). See images below, the pin size is the width in pixel of the blue box.

Tube image



Bar image



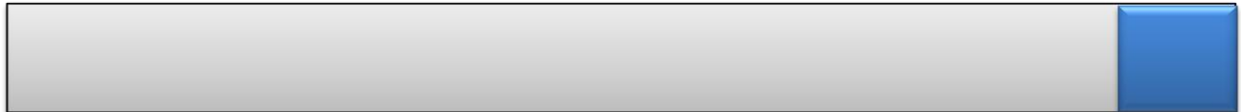
m = 0, value = 0



m = 0, value = 50



m = 0, value = 100



```
GUI BAR p  Pin_size
```

e.g.        GUI BAR p 50

Pin\_size: width or height (depends on orientation) of pin (in pixels)

## - BAR d

Disable bar at the start up.

```
GUI BAR d
```

### - BAR n

Name the bar, for easy identifying in software; user can specify the bar with a name.

```
GUI BAR n Name  
e.g.      GUI BAR n volume  
Name:     a text to be used as the name of bar
```

### - BAR E

End current percent bar setting.

```
GUI BAR E
```

## 6. SND Commands

This section is applicable when [gui\\_sound\\_plugin.c](#) is added to the project. The “SND” command is not the one with 5.15. “GUI SND” is a “GUI” command with a subcommand “SND”. Now it’s “SND” command and will be processed in the plug-in source code, not the GUI engine.

Unlike images or fonts, MP3 files are not stored on SDRAM. Only their paths are stored.

### 6.1 CFG: Configure global parameters.

#### - CFG D

Configure sound directory, default is “/” (root);

```
SND CFG D Path
e.g.      SND CFG D sr/snd
Path:     directory contains MP3 files
```

### 6.2 RG0: Register MP3 files.

```
SND RG0 ID nSegment Volume Lang Path
e.g.      SND RG0 0  1 80 255 start.mp3
           SND RG0 1 10 80 255 numbers.mp3

ID:        sound ID
nSegment:  segments in sound file, this is 1 for general files playing
           all at once, but can be more then one for segmented files
           For example; the numbers.mp3 sounds like "one two three ...
           zero", each segment can be played individually.
Volume:    sound volume, can be from 0-99.
Lang:      language for MP3 file
Path:      file path
```

### 6.3 RGC: Register MP3 files for multi-language application.

Note that files can be assigned to an ID if their contents are the same but in multiple languages. Software will play the right file (with the language mode at a time) automatically.

```

SND RGC Lang Path
e.g.    SND RG0 0 1 80 2 start_eng.mp3
        SND RGC          1 start_th.mp3

Lang:    language for MP3 file
Path:    file path

```

## 7. Writing Script

### 7.1 Script Structure

After that you know commands. Now let's see how to write your own script. A script file should have a head and body parts. The head part contains registering common objects to be used in the body part. The word common objects, whether they are used only one time or more, needed to be registered here. Then the body part describes what will be contains on screens and where they are.

#### - Head Part

Objects to be registered here are font, extension font, background image, image and sound. Notice that these objects are memory-consuming. So put them here and then use them again and again. Ordering here is no needed since each object has an assigned ID. And each register function has its own ID domain. At the end of this part, memory consumed is summarized so show it with “GUI MEM” command.

```
##HEAD##
//register font(s)
GUI RFONT 0 ...
GUI RFONT 1 ...
...
//register extension font(s)
GUI RXFONT 0 ...
GUI RXFONT 1 ...
...
//register background image(s)
GUI RBG 0 ...
GUI RBG 1 ...
...
//register image(s)
GUI RIMG 0 ...
GUI RIMG 1 ...
...
SND CFG D ...
SND RG0 0 ...
SND RGC 0 ...
SND RG0 1 ...
SND RGC 1 ...
...
//show memory used
GUI MEM
```

## - Body Part

The body part is segmented into screens. Begin each screen with “GUI SCS S” set some screen parameters if needed and then create buttons, tables or more as designed. End the script with “GUI END” so the engine shows how many objects created. Note that objects created here usually take many lines, that’s why we need “GUI x E” lines.

```
##BODY##
#####screen 0#####
GUI SCS S 0
GUI SCS C ...
GUI SND ...
GUI BG A ...

GUI BT ...
...
GUI BT E

GUI BT ...
...
GUI BT E

GUI LBL ...
...
GUI LBL E

...

#####screen 1#####
GUI SCS S 1
GUI SND ...
GUI BG A ...

GUI BT ...
...
GUI BT E

GUI TAB ...
...
GUI TAB E

...

#####screen 2#####
...

GUI END
```

## 7.2 Script Making Guidelines.

Actually the screen ordering is not needed. You can even have screen 0, 1 and then 3. Also you can put a button, a table and then a button again if you like. However, making script should be done tidily so the one who takes care of C programming can understand it well. It's recommend that script file should be made as follow:

1. Place same type objects together for easier reading.
2. Name screens and special buttons (buttons that stimulate some events). Buttons with name can be indentified easily. Buttons with functional scripts (key buttons) may be left unnamed since user has nothing to do with them.
3. Utilize auxiliary values. Auxiliary value can be assigned to screens and buttons, utilizing them make the code more flexible as the value can be changed easily.
4. Use “save” and “load” for similar buttons. If a screen has buttons sharing some parameters, “save” and “load” reduce script lines and let you change parameters once for all.
5. Left parameters with default value unset (for example; black text, white background). Also reserve ID 0 for most common font. These save many of script lines.

## 8. GUI Engine Setup Guide

This section describes how to start a project with GUI script.

### 8.1 Download Example Project

First of all, download latest example project from our website. With the project, all necessary source files needed by GUI engine library are ready. Anyway, the library itself needs to be purchased and will be sent to you by email.

Since the GUI Script version 1.09 and later, example project and library for Yagarto is freely downloadable. It's originally compiled with the following version of Yagarto (see figure below). For newer version, the makefile may need to be modified, for example, **change from "arm-elf-gcc" to "arm-none-eabi-gcc"**. See more information about how to get and install Eclipse IDE and Yagarto compiler on the first section of appendix.

```
arm-elf-gcc (GCC) 4.3.3
Copyright (C) 2008 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

### 8.2 Add GUI Engine Library

Depends on compiler, making the library available for the example project are different.

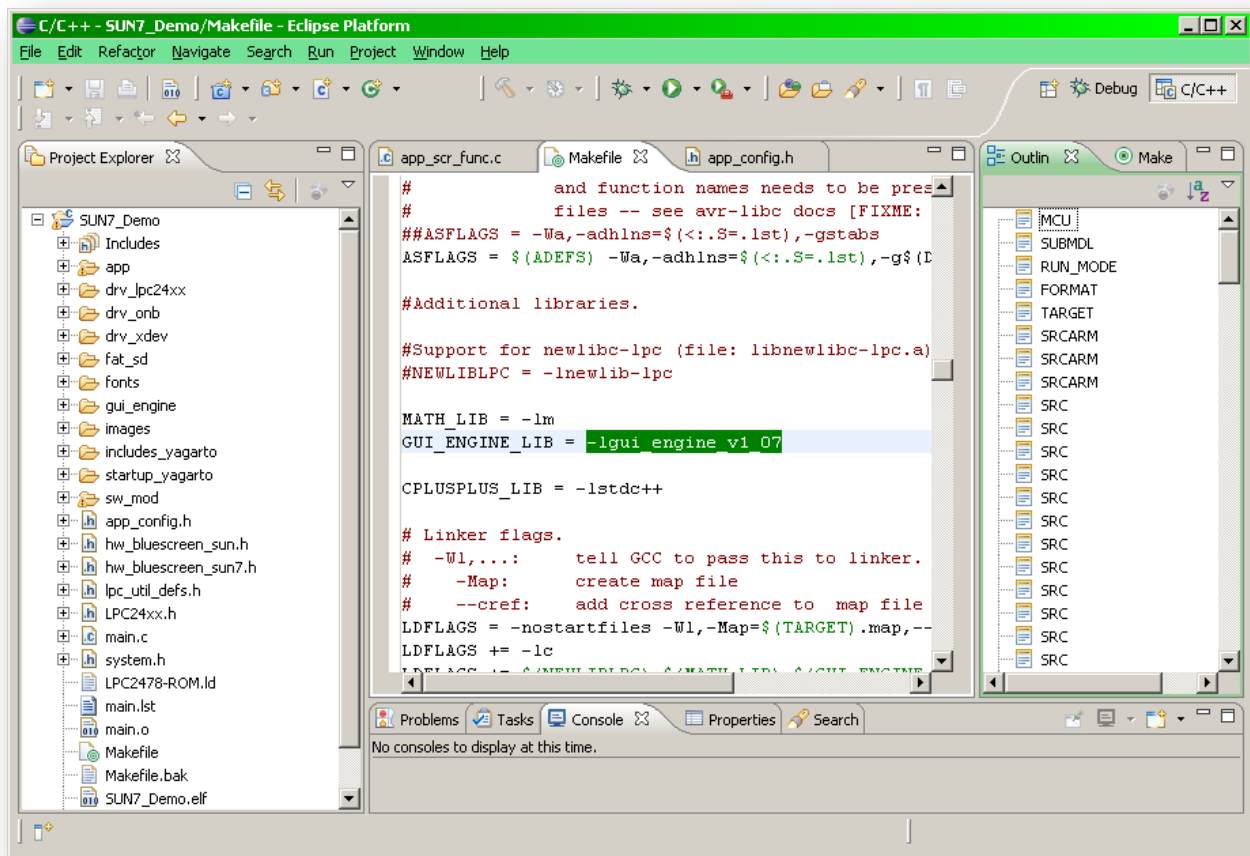
- **Yagarto**: Put the library file (.a file) in the library path, for example:

C:\Program Files\yagarto\arm-elf\lib

Note that the "arm-elf" could be different among versions of Yagarto. In such case, the makefile must be modified.

To upgrade, the library name in the makefile must be changed to the newer one (see figure below). Note that the "-lgui\_engine\_v1\_07" is the option that forces the linker to search symbols from the "libgui\_engine\_v1\_07.a" file placed in the library path.



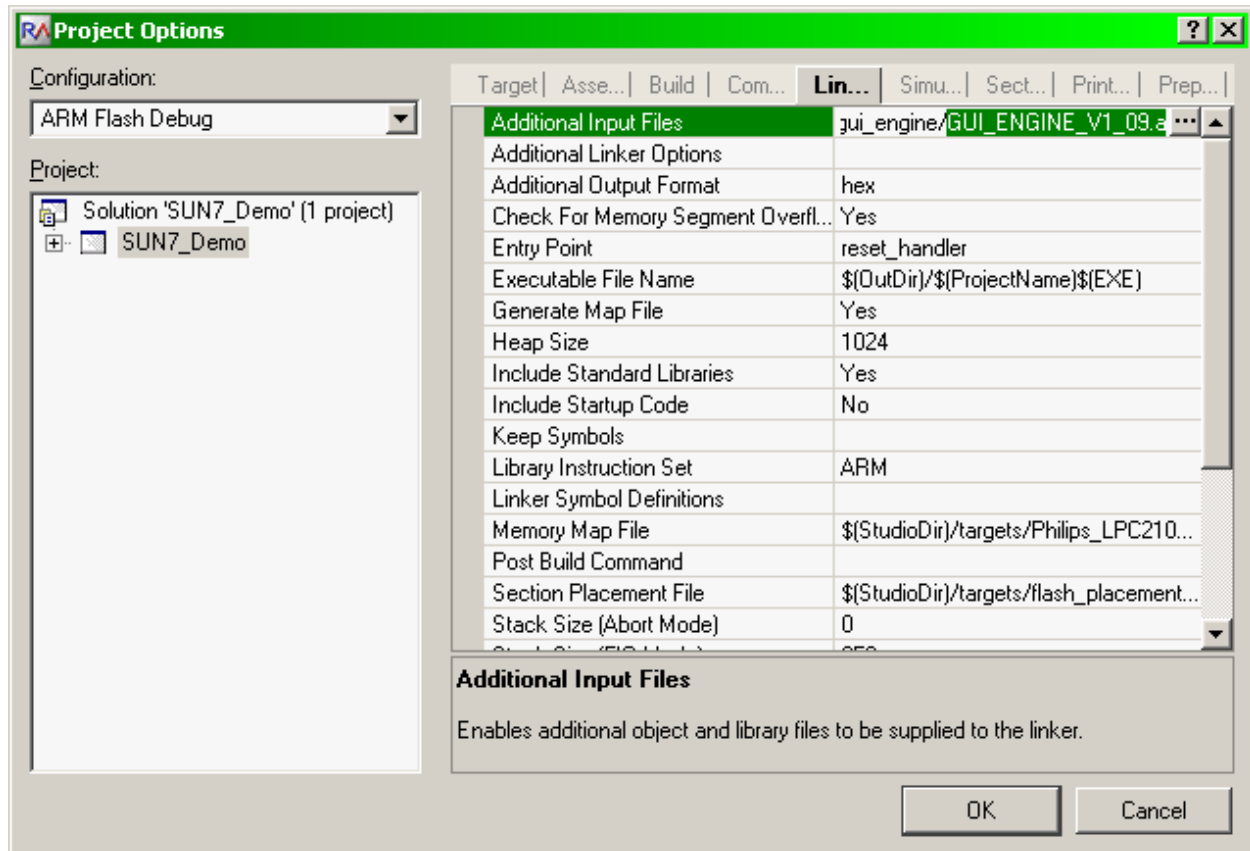


- **uVision**: Put the library in the library path, for example:

C:\Keil\ARM\RV31\LIB.

To upgrade, put new library in the library path, remove the old library from the project window and then add the new one.

- **CrossWorks**: Put the library file in the project and locate it in the Project Options>Linker>Additional Input Files as shown below.



**-IAR Embedded Workbench IDE:** The library can be added directly to the project.

### 8.3 Disabling Sound Plug-in

To disable sound plug-in (for quite applications), comment out a line in [app\\_config.h](#).

```
//#include "gui_engine/gui_sound_plugin_v2_00.h"
```

And the I/O pin functions connected to MP3 decoder IC are selected in [main.c](#), comment them out for ones who need them to be GPIO.

```
//PINSEL5 |= (3 << 12); //SCLK0
//PINSEL5 |= (3 << 20); //MISO0
//PINSEL5 |= (3 << 22); //MOSI0
```

## 9. Application Programming Interface

This section provides information on how to make application based on the GUI script. First of all, users should specify quantities for objects. So the GUI engine can provide enough memory for them. After that, creates their own functions and assign them to events.

### 9.1 Setting Parameters

Images, buttons and other objects need memory to store their parameters. Users must specify them first so there is enough memory reserved. At the final state, when the new firmware revision is released, **users should specify quantities so more objects can be added (from script) in the future.**

Specifying should be done in `app_scr_func.h`. Here default values are set. Note that background images use constant size of memory, whether they are assigned in script or not.

```
//a part from app_scr_func.h
#define MAX_SCR_OBJ          40
#define MAX_SCR              140          //all screens
#define MAX_BACKGROUND      10          //all background available
#define MAX_IMAGE           400          //all images
#define MAX_BUTTON          400          //all buttons
#define MAX_IMAGE_BOX       400          //all image boxed
...
```

Table below describes parameters specifiable; some of them have default value and don't need to be set.

Parameters	Instance	Description
MAX_SCR_OBJ	so_obj	Reserved dynamic objects on a screen (see more detail about dynamic object later in this section)
MAX_SCR	guisc	Reserved screen slots
MAX_BACKGROUND	guibg	Reserved background image slots
MAX_IMAGE	guiimage	Reserved image slots used for buttons and image boxes
MAX_BUTTON	guibt	Reserved button slots
MAX_IMAGE_BOX	guiimgbox	Reserved image box slots
MAX_TEXTBOX	guitxt	Reserved textbox slots
MAX_TABLE	guitab	Reserved table slots
MAX_TEXT_SIZE	guiltxt	Reserved text pool size
MAX_LABEL	guilbl	Reserved label slots
MAX_BOX	guibox	Reserved box slots
MAX_PERCENT_BAR	guibar	Reserved percent bar slots
MAX_FONT	guifont	Reserved font slots
MAX_EXT_FONT	guixfont	Reserved extension font slots
MAX_IMG_FONT	guiifont	Reserved image font slots
MAX_SOUND_FILE	snd file	Reserved sound file slots
MAX_SOUND	snd	Reserved sound slots
MAX_SOUND_SEQ	sndlist	Longest sound sequence on a screen

**Remark:**    -All parameters must to be set to 1 or more  
                   -Sound related parameters have to be set only when sound plug-in is used

On a screen, not every object is dynamic. Buttons, tables and textboxes have something to do when they are pressed, so they are dynamic. In the same time, labels, image boxes and boxes are static. Only dynamic objects are counted as objects on a screen and must not be over than MAX\_SCR\_OBJ.

Texts and names set with scripts are continuously concatenated in a text pool. Even a textbox having length set, will has memory reserved for its text here. All of these texts and names size together must not be over than MAX\_TEXT\_SIZE.

Besides of specifying numbers of objects, user can configure some settings in [app\\_scr\\_func.h](#), table below shows some parameters that can be set for appropriate operation.

Parameters	Definition	Description
SHOW_ACTIVITY	Define it as 0 or 1	Show activities on LCD while the script being read, left it undefined here to use default value 1 (show)
EMBED_SCRIPT_TO_ROM	Define it as 0 or 1	Embed script (main.txt only) into FLASH memory for easier development process, file2h.exe needed to convert script file to header file default value 0 (not embed, use file from SD card originally)

Using EMBED\_SCRIPT\_TO\_ROM as 1, file2h.exe is needed. This software is in the example folder downloadable from our website. The user has to run it on Window's console or making a batch file to run it.

Anyway, with library limitation, some parameters are used as array sizes and must be fixed to the same as in the library. So users **must not change these values**.

//Parameters below are not to be changed, they must match the GUI Script library

```
#define MAX_LANGUAGE      8 //can not be more than 8
#define MAX_PATH          8
#define MAX_TABLE_COL     20
#define AUX_INT_NUM       4
#define AUX_CHAR_NUM       4
```

## 9.2 Object Structures

After script reading done, all parameters are stored in object structures. Here users can get and set them directly from application code. Some components of structures are functions; these are events that users can bind them to their own functions on purpose. All structures are declared in [gui\\_engine.h](#) (and [gui\\_sound\\_plugin.h](#) for sound related structures).

To understand better, characters in “RW” column are described below;

RW	Description
R	Parameters usually be read by users on purpose
W	Parameters usually be written by users on purpose
RW	Parameters usually be read and written by users on purpose
-	For general use, there is no need for users to read or write parameters

It's true that all parameters can be read by software with no corrupt. And with some objects that users intend to set their parameters by software (for example, a line: “BT E” creates a bare button), all parameters can be written. The “RW” columns here are only for general use that parameters are set from script file.

**Type: scr\_t, instance: guisc[] => Parameters associated with screens**

Type	Name	Description	RW
unsigned short	bg [MAX_LANGUAGE]	background image IDs	-
unsigned short	bt_start	first button ID associated	-
unsigned short	bt_num	total buttons associated	-
unsigned short	txt_start	first textbox ID associated	-
unsigned short	txt_num	total textboxes associated	-
unsigned short	tab_start	first table ID associated	-
unsigned short	tab_num	total tables associated	-
unsigned short	lbl_start	first label ID associated	-
unsigned short	lbl_num	total labels associated	-
unsigned short	box_start	first box ID associated	-
unsigned short	box_num	total boxes associated	-
unsigned short	img_start	first images ID associated	-
unsigned short	img_num	total images associated	-
unsigned short	bar_start	first bar ID associated	-
unsigned short	bar_num	total bars associated	-
unsigned short	snd	sound ID from "snd" to be played	-
unsigned short	path	target path when the screen displayed	-
unsigned int	wait_time	wait time in 100ms	-

unsigned short	goto_scr [MAX_PATH]	target screen id when timeout with wait-and-go set	-
long	aux_32 [AUX_INT_NUM]	auxiliary integer-type variable(s)	R
char	aux_8 [AUX_CHAR_NUM]	auxiliary character-type variable(s)	R
unsigned long	color	background color in case of plane background used	-
unsigned short	set bit0-2 bit3 bit4 bit5 bit6 bit7-9 bit10 bit11 bit12 bit13-15	internal use only: background added 1 = plain background set 1 = wait-and-go set 1 = play sound at start 1 = screen name set keypad on screen change path set extension script added 1 = wait-and-close (popup) reserved	-
unsigned short	hsize	horizontal size (popup)	-
unsigned short	vsize	vertical size (popup)	-
char *	name	stores address of screen name	R
void *	init	event before the screen drawing	W
void *	init2	event after the screen drawing	W
void *	task100ms	event every 100ms while the screen is showing	W
void *	kp_char_event	event when keypad is pressed	W
void *	txtk_char_ event	event when new character passed to key textbox, occur before writing textbox	W
void *	snd_end	event at sound end	W
char *	xscript	store address of extension filename	-

**Type: back\_t, instance: guibg[] => Parameters associated with background images**

Type	Name	Description	RW
unsigned char	lang_mode	language mode : 0xFF = all, (1 << n) = language n, default = all	-

**Type: image\_t, instance: guiimage[] => Parameters associated with images**

Type	Name	Description	RW
unsigned short	hsize	horizontal size	-
unsigned short	vsize	vertical size	-
unsigned char	lang_mode	language mode : 0xFF = all, (1 << n) = language n, default = all	-
unsigned char	set bit0 bit1-7	internal use only : color button reserved	-
unsigned long	address	address of image in memory, color value for color buttons	-
unsigned long	border color	border color for color buttons	-



**Type: button\_t, instance: guibt[] => Parameters associated with buttons**

Type	Name	Description	RW
unsigned short	horigin	horizontal origin	-
unsigned short	vorigin	vertical origin	-
unsigned char	hmove	rightward shift distance when pressed	-
unsigned char	vmove	downward shift distance when pressed	-
unsigned char	cmd	special command	-
int	aux_val	auxiliary value for general purpose, for example, used as a number of keypad	R
unsigned short	disa_image	image id for disable state	-
unsigned short	norm_image	image id for normal state	-
unsigned short	pres_image	image id for pressed state	-
unsigned short	goto_scr [MAX_PATH]	screen to go when button is released	-
void *	do_	event every case the button associated	W
void *	press	function done when button is pressed	W
void *	release	function done when button is released	W
char *	text	stores address of text	-
char *	name	stores address of name	R
unsigned long	text_color	text color, default = black	-
unsigned short	font_id	font id	-
unsigned short	xfont_id	extension font id (for example: Thai font)	-
char *	xscript	store address of extension filename	-
unsigned long	set bit0 bit1 bit2 bit3 bit4 bit5  bit6  bit7 bit8  bit9 bit10-13	internal use only : disa_image set norm_image set pres_image set goto set 0 = enable, 1 = disable 0 = full display, 1 = transparent 0 = image button, 1 = color button text on button set bold characters (bit7 must be set) extension font set alignment, 0 = left, 1 = center, 2 = right ,default = 1 name set	-

	bit14	inverse when pressed	
	bit15	interlock mode set	
	bit16	interlock order (0-7)	
	bit17-19	repeat	
	bit20	extension script added	
	bit21	open popup window	
	bit22	close popup window	
	bit23	reserved	
	bit24-30	skip this button	
	bit31		

**Type: image\_box\_t, instance: guimgbox[] => Parameters associated with image boxes**

Type	Name	Description	RW
unsigned short	horigin	horizontal origin	-
unsigned short	vorigin	vertical origin	-
unsigned short	image_id	image id	-
char *	name	stores address of name	R
unsigned short	set bit0-2 bit3 bit4 bit5  bit6 bit7	internal use only : reserved name set 0 = enable, 1 = disable 0 = full display, 1 = transparent 0 = image box, 1 = color box reserved	-

**Type: textbox\_t, instance: guitxt[] => Parameters associated with textboxes**

Type	Name	Description	RW
unsigned short	horigin	horizontal origin	-
unsigned short	vorigin	vertical origin	-
unsigned short	hsize	horizontal size	-
unsigned short	vsize	vertical size	-
unsigned long	back_color	background color, default = white	-
unsigned long	text_color	text color, default = black	-
unsigned long	border_color	border color	-
unsigned short	max_len	maximum characters	- [1]
unsigned char	pwd_ch	password character	-
char *	text	occupied string	RW
unsigned short	font_id	font id	-
unsigned short	xfont_id	extension font id (for example: Thai font)	-
char *	name	stores address of name	R
unsigned short	set bit0-3	internal use only : alignment, 0 = left, 1 = center, 2 = right	-
	bit4	insertable set (set cursor on)	
	bit5	receiver from key buttons	
	bit6	bold characters	
	bit7	font_id set	
	bit8	border on	
	bit9	password mode set	
	bit10	extension font set	
	bit11	image font set	
	bit12	multi-line set	
	bit13	reserved	
	bit14	name set	
	bit15	skip this textbox	

[1] .text is initiated in the text pool with .max\_len size from script, .max\_len must not be changed by software to bigger size

**Type: table\_t, instance: guitab[] => Parameters associated with tables**

Type	Name	Description	RW
unsigned short	horigin	horizontal origin	-
unsigned short	vorigin	vertical origin	-
unsigned long	text_color	text color, default = black	-
unsigned long	border_color	border color, default = black	-
unsigned short	col_width [MAX_TABLE_ COL]	column width	-
char *	col_text [MAX_TABLE_ COL]	text in columns on the first row (captions)	-
unsigned short	col_capt_color [MAX_TABLE_ COL]	column's caption color, default = white	-
unsigned short	col_cont_color [MAX_TABLE_ COL]	column's content color, default = white	-
unsigned short	row height	height of each row	-
unsigned short	row_num	total row	-
unsigned short	font_id	font id	-
unsigned short	xfont_id	ext font id (for example: Thai font)	-
void *	do_	event every case the table associated	W
void *	press	function done when table pressed	W
char *	name	stores address of name	R
unsigned char	set bit0-3  bit4 bit5 bit6 bit7	internal use only : alignment, 0 = left, 1 = center, 2 = right bold characters extension font set name set skip this table	-

**Type: percent\_bar\_t, instance: guibar[] => Parameters associated with percent bars**

Type	Name	Description	RW
unsigned short	horigin	horizontal origin	-
unsigned short	vorigin	vertical origin	-
unsigned short	image_id	image id	-
unsigned short	pin_size	pin size	-
unsigned char	val	value in percent(0-100)	RW
unsigned char	sens	sensitivity applied only when touch response style is 3	-
unsigned short	last_gpos	last position, internal use only	-
char *	name	stores address of name	R
void *	move	event while moving (every 10ms)	W
void *	stop	event when moving stopped	W
unsigned char	set bit0-2	internal use only : touch response style, 0 = none, 1 = move to the touch, 2 = follow the touch movement, 3 = slide with touch movement	-
	bit3	orientation, 0 = horizontal, 1 = vertical	
	bit4	0 = enable, 1 = disable	
	bit5	name set	
	bit6	0 = image bar, 1 = colored bar	
	bit7	skip this bar	

**Type: box\_t, instance: guibox[] => Parameters associated with boxes**

Type	Name	Description	RW
unsigned short	horigin	horizontal origin	-
unsigned short	vorigin	vertical origin	-
unsigned short	hsize	horizontal size	-
unsigned short	vsize	vertical size	-
unsigned long	color	text color, default = black	-

**Type: label\_t, instance: guilbl[] => Parameters associated with labels**

Type	Name	Description	RW
unsigned short	horigin	horizontal origin	-
unsigned short	vorigin	vertical origin	-
unsigned long	color	text color, default = black	-
unsigned short	font_id	font id	-
unsigned short	xfont_id	extension font id (for example: Thai font)	-
char *	text	string pointer	RW
unsigned char	lang_mode	language selection : 0xFF = all, (1 << n) = language n, default = all	-
char *	name	stores address of name	R
unsigned short	set bit0-3	internal use only : alignment, 0 = left, 1 = center, 2 = right	-
	bit4	bold characters	
	bit5	extension font set	
	bit6	image font set	
	bit7	skip this label	
	bit8	name set	
	bit9-15	reserved	

**Type: font\_t, instance: guifont[] => Parameters associated with fonts**

Type	Name	Description	RW
unsigned long	address	address of font in memory	-
unsigned short	width	font width	-
unsigned short	height	font height	-
unsigned short	gap	gap between characters	-

**Type: xfont\_t, instance: guixfont[] => Parameters associated with extension fonts**

Type	Name	Description	RW
unsigned long	address	address of font in memory	-
unsigned short	width	font width	-
unsigned short	height	font height	-
unsigned short	adj	different height between English font and extension font	-
unsigned char	aux_8	auxiliary value used differently by languages	-

**Type: ifont\_t, instance: guiifont[] => Parameters associated with image fonts**

Type	Name	Description	RW
unsigned char	char1	first character	-
unsigned short	nchar	number of characters	-
unsigned short	image1	first image ID	-
unsigned short	gap	gap between characters	-

Sound related structures from [gui\\_sound\\_plugin.h](#).

**Type: sound\_file\_t, instance: snd\_file[] => Parameters associated with sound files**

Type	Name	Description	RW
unsigned char	lang_mode	language selection : 0xFF = all, (1 << n) = language n, default = all	-
char *	name	file name referenced from snd_dir	-

**Type: sound\_t, instance: snd[] => Parameters associated with sounds**

Type	Name	Description	RW
unsigned short	file [MAX LANGUAGE]	files from snd_file[]	-
unsigned char	segments	how many segments in a file, default = 1	-
unsigned char	vol	sound volume:0-100, default = 50	-

**Type: sound\_list\_t, instance: sndlist[] => Parameters associated with sound playlist**

Type	Name	Description	RW
unsigned short	snd_no	sound number, start with 0	-
unsigned char	segment_no	segment number, start with 0	-

## 9.3 API Functions (GUI)

### - Main GUI functions

Functions in this section are called by system and users don't need to use.

- **GUIInit**: Initialize objects to default value. Used once before reading script. This is already placed in example project in [app\\_bs\\_sun\\_demo.c](#).

```
void GUIInit(unsigned char report_on)
report_on: use 1 will allow error report on console
```

- **GUIReadCmd**: Function processes when GUI commands found. Used from [app\\_console.c](#).

```
int GUIReadCmd(unsigned int *cons_i)

cons_i:    index of console buffer to be processed

return:    wrong argument, 0 means no error
```

- **GUISkipFileLoading**: On the development when users have script unchanged and users are modifying their code, it may take quite long time when board is reprogrammed and boot. Use this function before ReadScript called, it makes script reading faster by skipping files loading (images, background, fonts). Beware to use this only when the board is always powered with all files loaded into SDRAM once and all scripts are unchanged (program the board once with this function commented out and uncomment it for next programming without removing power). Note that there may be losses of data on SDRAM, these only occur with images and fonts but not with object's parameters.

```
void GUISkipFileLoading(void)
```

- **GUIGetFreeAddress**: Use this function to get free memory start address in SDRAM after script reading completed. The rest of memory can be used for general purpose. Also mind that there is reserved memory for JPEG decoding at the end of SDRAM.

```
unsigned long GUIGetFreeAddress(void)
```

## - Font-related functions

- **GUICfgFont**: Configure font from font\_id, use xfont\_set if extension font used.

```
void GUICfgFont(unsigned short font_id,
                unsigned short xfont_id, unsigned long bold,
                unsigned long xfont_set)

font_id:    ID of font
xfont_id:    ID of extension font
bold:        use 1 for bold text
xfont_set:   use 1 if extension font needed to be configured or use 0
              and xfont_id will be ignored
```



## - Language-related functions

- **GUISetLang**: change language mode.

```
void GUISetLang(unsigned char lang)
lang:      language mode, must be set in 1<<n where n:0-7
```

- **GUIGetLang**: Get language mode.

```
unsigned char GUIGetLang(void)
return:     language mode
```

## - Screen-related functions

- **GUIGetScrID**: Find screen ID by their names.

```
int GUIGetScrID(char *scr_name)
scr_name:   screen name
return:     screen ID or -1 when not found
```

- **GUIScreenInit**: Initialize components in screen, it's called every time that screens change. User init functions (.init) are called here inside GUIScreenInit. Initializations here could be enabling, disabling, assignment or copying string to textbox and mores but without display-related functions.

```
void GUIScreenInit(void)
```

- **GUIScreenInit2**: Initialize components in screen after the background and objects are shown. Initializations here could be adding text to textboxes, writing text in table contents, showing images and etc.

```
void GUIScreenInit2(void)
```

- **GUIGotoScreen**: Change to target screen.

```
void GUIGotoScreen(unsigned short _new_scr_id)
_new_scr_id: target screen ID
```

- **GUIOpenPopup**: Open popup screen.

```
void GUIOpenPopup(unsigned short _popup_scr_id)
_popup_scr_id: target screen ID
```

- **GUIClosePopup**: Close popup screen.

```
void GUIClosePopup(void)
```

## - Keypad-related functions

- **GUISetKeypadNo**: Set keypad number. Have the same effect as “SCS K”.

```
void GUISetKeypadNo(unsigned short _scr_id,
unsigned char _kp_no)
_scr_id: screen ID
_kp_no: keypad type (see command “SCS K”)
```

- **GUIBindCustomKP3Init**: When use customized keypad (kp3), users need to bind their keypad init function to GUI engine. Bound function will be called when screens that use customized keypad are shown (screen with “SCS K 6” or “SCS K 7”). Binding (using GUIBindCustomKP3Init) should be done in AppScrInit().

```
void GUIBindCustomKP3Init(void (*func)
(unsigned char kp_no, unsigned char obj_no,
kp3_t *kp))
func: customized keypad initial function
```

## - Button-related functions

- **GUIEnableBt**: Enable buttons. If the button is shown, this will not redraw the button; follow this function by GUIRefreshBt to redraw with enable state. Enabling button in screen init function (.init) doesn’t need GUIRefreshBt.

```
void GUIEnableBt(unsigned short id)
id: button ID
```

- **GUIDisableBt**: Disable buttons. If the button is shown, this will not redraw the button; follow this function by GUIRefreshBt to redraw with disable state. Disabling button in screen init function (.init) doesn't need GUIRefreshBt.

```
void GUIDisableBt(unsigned short id)

id:      button ID
```

- **GUIGetBtID**: To enable, disable or read some parameters from buttons, its ID must be known. But from the script, buttons are created and assigned ID by their order. For example; the first button of the first screen will have ID 0 and the first button of the second screen will have ID equal to the quantity of button in the first screen. So with known screen ID and the button order in the screen it belongs to, use GUIGetBtID function to get its ID.

```
unsigned short GUIGetBtID(unsigned short _scr_id,
                          unsigned short _bt_no)

_scr_id:  screen ID
_bt_no:   button order in the screen it belongs to

return:   button ID
```

- **GUIGetBtID2**: An alternative way to get a button ID by its name. The button name must first be set from script with "BT n".

```
int GUIGetBtID2(unsigned short _scr_id, char *bt_name)

_scr_id:  screen ID
bt_name:  button name

return:   button ID, -1 when not found
```

- **GUIGetBtID3**: Similar to GUIGetBtID2 but with screen name, not screen ID.

```
int GUIGetBtID3(char *scr_name, char *bt_name)

scr_name:  screen name
bt_name:   button name

return:    button ID, -1 when not found
```

- **GUIRefreshBt**: Use to redraw button. This function is assumed to be used with current screen, so it doesn't need screen ID parameter.

```
void GUIRefreshBt(unsigned short _bt_no)
_bt_no:    button order in the screen it belongs to
```

- **GUIRefreshBt2**: Similar to GUIRefreshBt but target button is identified by its name.

```
void GUIRefreshBt2(char *bt_name)
bt_name:    button name
```

- **GUISkipBt**: Use this function with buttons to skip them out. Not just disable them, the buttons will not even be shown on screen. Use GUIUnskipBt to cancel the effect.

```
void GUISkipBt(unsigned short id)
id:         button ID
```

- **GUIUnskipBt**: Use this function with buttons to cancel the effect of GUISkipBt.

```
void GUIUnskipBt(unsigned short id)
id:         button ID
```

## - Textbox-related functions

- **GUIGetTxtID**: Get textbox ID, similar to GUIGetBtID but with textbox type objects. See GUIGetBtID for more detail.

```
unsigned short GUIGetTxtID(unsigned short _scr_id,
                           unsigned short _txt_no)
_scr_id:    screen ID
_txt_no:    textbox order in the screen it belongs to
return:     textbox ID
```

- **GUIGetTxtID2**: Get textbox ID from its name, similar to GUIGetBtID2. The textbox name must first be set from script with “TXT n”.

```
int GUIGetTxtID2(unsigned short _scr_id, char *txt_name)

_scr_id:    screen ID
txt_name:   textbox name

return:     textbox ID, -1 when not found
```

- **GUIGetTxtID3**: Get textbox ID from its name and screen name, similar to GUIGetBtID3.

```
int GUIGetTxtID3(char *scr_name, char *txt_name)

scr_name:   screen name
txt_name:   textbox name

return:     textbox ID, -1 when not found
```

- **GUIGetTxtObj**: Get textbox object number. Unlike textbox ID, object number tells us the order of objects in a screen used to with so\_obj[]. This function is used mostly by the GUI engine itself. In general case, users can ignore this function.

```
unsigned short GUIGetTxtObj(unsigned short _scr_id,
                           unsigned short _txt_no)

_scr_id:    screen ID
_txt_no:    textbox order in the screen it belongs to

return:     textbox object number
```

- **GUIAddTxt**: For textbox that is just displayed on a screen, its text can be simply assigned by strcpy or other functions. Anyway, for key textbox that has to be initialized on the screen init (for example; username setting screen, old username has to be displayed on init), use GUIAddTxt so cursor and position of new character are correct.

```
void GUIAddTxt(unsigned short _scr_id,
               unsigned short _txt_no, char *str)

_scr_id:    screen ID
_txt_no:    textbox order in the screen it belongs to
str:        string to be added
```

- **GUIAddTxt2**: Similar to GUIAddTxt, but target textbox is identified by its name.

```
void GUIAddTxt2(unsigned short _scr_id,
char *txt_name, char *str)

_scr_id:    screen name
txt_name:   textbox name
str:        string to be added
```

- **GUIAddTxt3**: Similar to GUIAddTxt, but target screen and target textbox are identified by their names.

```
void GUIAddTxt3(char *scr_name, char *txt_name, char *str)

scr_name:   screen ID
txt_name:   textbox name
str:        string to be added
```

- **GUIClrTxt**: With the same reason with GUIAddTxt, this is the one to be used to clear key textboxes.

```
void GUIClrTxt(unsigned short _scr_id,
unsigned short _txt_no)

_scr_id:    screen ID
_txt_no:    textbox order in the screen it belongs to
```

- **GUIClrTxt2**: Similar to GUIClrTxt, but target textbox is identified by its name.

```
void GUIClrTxt2(unsigned short _scr_id, char *txt_name)

_scr_id:    screen ID
txt_name:   textbox name
```

- **GUIClrTxt3**: Similar to GUIClrTxt, but target screen and target textbox are identified by their names.

```
void GUIClrTxt3(char *scr_name, char *txt_name)

scr_name:   screen name
_txt_no:    textbox name
```

- **GUIRefreshTxt**: Refresh textbox. After textbox shown, GUIAddTxt and GUIClrTxt won't update the screen. Follow them by GUIRefreshTxt to update textboxes. This function is assumed to be used with current screen, so it doesn't need screen ID parameter.

```
void GUIRefreshTxt(unsigned short _txt_no)
txt_no:    textbox order in the screen it belongs to
```

- **GUIRefreshTxt2**: Similar to GUIRefreshTxt, but target textbox is identified by its name.

```
void GUIRefreshTxt2(char *txt_name)
txt_name:  textbox name
```

- **GUISetTxtKey**: In some application, many key textboxes (textbox that receives characters from keypad, key buttons) may be used on one screen. Only one textbox can be active at a time. In such case, this function can be used to change key textbox.

```
void GUISetTxtKey(unsigned short _scr_id,
                  unsigned short _txt_no)
_scr_id:    screen ID
_txt_no:    textbox order in the screen it belongs to
```

- **GUIClrTxtKey**: Prior to set key textbox, use this function to clear current key textbox.

```
void GUIClrTxtKey(void)
```

- **GUISkipTxt**: Use this function with textboxes to skip them out and they will not be shown on screen. Use GUIUnskipTxt to cancel the effect.

```
void GUISkipTxt(unsigned short id)
id:           textbox ID
```

- **GUIUnskipTxt**: Use this function with textboxes to cancel the effect of GUISkipTxt.

```
void GUIUnskipTxt(unsigned short id)
id:          textbox ID
```

- **GUIInsertChar**: Use this function to insert a character to the key textbox.

```
void GUIInsertChar(char ch)
ch:          character
```

## - Label-related functions

- **GUIPrintLbl**: Print label on the screen. This is used by GUI engine itself. In general case, users can ignore this function.

```
void GUIPrintLbl(unsigned short id)
id:          label ID
```

- **GUIGetLblID**: Get label ID, similar to GUIGetBtID.

```
unsigned short GUIGetLblID(unsigned short _scr_id,
                           unsigned short _lbl_no)
_scr_id:      screen ID
_lbl_no:      label order in the screen it belongs to
return:       label ID
```

- **GUIGetLblID2**: Get label ID from its name, similar to GUIGetBtID2. The label name must first be set from script with “LBL n”.

```
int GUIGetLblID2(unsigned short _scr_id, char *lbl_name)
_scr_id:      screen ID
lbl_name:     label name
return:       label ID, -1 when not found
```



- **GUIGetLblID3**: Get label ID from its name and screen name, similar to GUIGetBtID3.

```
int GUIGetLblID3(char *scr_name, char *lbl_name)

scr_name:  screen name
lbl_name:  label name

return:    label ID, -1 when not found
```

- **GUISkipLbl**: Use this function with labels to skip them out and they will not be shown on screen. Use GUIUnskipLbl to cancel the effect.

```
void GUISkipLbl(unsigned short id)

id:        label ID
```

- **GUIUnskipLbl**: Use this function with labels to cancel the effect of GUISkipLbl.

```
void GUIUnskipLbl(unsigned short id)

id:        label ID
```

## - Table-related functions

- **GUIGetTableID**: Get table ID, similar to GUIGetBtID.

```
unsigned short GUIGetTableID(unsigned short _scr_id,
                             unsigned short _tab_no)

_scr_id:    screen ID
_tab_no:    table order in the screen it belongs to

return:     table ID
```

- **GUIGetTableID2**: Get table ID from its name, similar to GUIGetBtID2. The table name must first be set from script with “TAB n”.

```
int GUIGetTableID2(unsigned short _scr_id, char *tab_name)

_scr_id:    screen ID
tab_name:   table name

return:     table ID, -1 when not found
```

- **GUIGetTableID3**: Get table ID from its name and screen name, similar to GUIGetBtID3.

```
int GUIGetTableID3(char *scr_name, char *tab_name)

scr_name:  screen name
tab_name:  button name

return:    table ID, -1 when not found
```

- **GUIWriteTable**: Write text in table with options defined by script.

```
void GUIWriteTable(unsigned short _tab_no,
                  unsigned short row, unsigned short col, char *text)

_tab_no:  table order in the screen it belongs to
row:      row number, start with 0
col:      column number, start with 0
text:     text to be written
```

- **GUIWriteTable2**: An extended version of GUIWriteTable. With this one, parameters can be specified beyond settings from script.

```
void GUIWriteTable2(unsigned short _tab_no,
                  unsigned short row, unsigned short col, char *text,
                  unsigned long back_color, unsigned long font_color)

_tab_no:  table order in the screen it belongs to
row:      row number, start with 0
col:      column number, start with 0
text:     text to be written
back_color: back color (cell color)
font_color: font color
```

- **GUISkipTable**: Use this function with tables to skip them out and they will not be shown on screen. Use GUIUnskipTable to cancel the effect.

```
void GUISkipTable(unsigned short id)

id:      table ID
```

- **GUIUnskipTable**: Use this function with tables to cancel the effect of GUISkipTable.

```
void GUIUnskipTable(unsigned short id)

id:      table ID
```

## - Image box-related functions

- **GUIShowImg**: Show images on the screen. Usually follows GUIEnableImg.

```
void GUIShowImg(unsigned short id)

id:      image ID
```

- **GUIGetImgID**: Get image ID, similar to GUIGetBtID.

```
unsigned short GUIGetImgID(unsigned short _scr_id,
                           unsigned short _img_no)

_scr_id:  screen ID
_img_no:  image order in the screen it belongs to

return:   image ID
```

- **GUIGetImgID2**: Get image ID from its name, similar to GUIGetBtID2. The image name must first be set from script with “IMG n”.

```
int GUIGetImgID2(unsigned short _scr_id, char *img_name)

_scr_id:  screen ID
img_name: image name

return:   image ID, -1 when not found
```

- **GUIGetImgID3**: Get image ID from its name and screen name, similar to GUIGetBtID3.

```
int GUIGetBtID3(char *scr_name, char *img_name)

scr_name: screen name
img_name: image name

return:   image ID, -1 when not found
```

- **GUIEnableImg**: Enable image, similar to GUIEnableBt. This may need GUIShowImg to show enabled images, unless using it in screen init function (.init).

```
void GUIEnableImg(unsigned short id)
id:      image ID
```

- **GUIDisableImg**: Disable image, unlike buttons, disabled images are not shown on the screen. Currently, removing images from the screen is not supported. An image may be needed to cover old one. This function is useful when use in .init function.

```
void GUIDisableImg(unsigned short id)
id:      image ID
```

## - Percent bar-related functions

- **GUIRefreshBar**: Redraw bar after its value change by software. To change the bar's value, the user can directly set .val.

```
void GUIRefreshBar(unsigned short id)
id:      bar ID
```

- **GUIGetBarID**: Get bar ID, similar to GUIGetBtID.

```
unsigned short GUIGetBarID(unsigned short _scr_id,
                           unsigned short _bar_no)
_scr_id:    screen ID
_bar_no:    bar order in the screen it belongs to
return:     bar ID
```

- **GUIGetBarID2**: Get bar ID from its name, similar to GUIGetBtID2. The bar name must first be set from script with “BAR n”.

```
int GUIGetBarID2(unsigned short _scr_id, char *bar_name)

_scr_id:    screen ID
bar_name:   bar name

return:     bar ID, -1 when not found
```

- **GUIGetBarID3**: Get bar ID from its name and screen name, similar to GUIGetBtID3.

```
int GUIGetBarID3(char *scr_name, char *bar_name)

scr_name:   screen name
bar_name:   bar name

return:     bar ID, -1 when not found
```

- **GUIEnableBar**: Enable bar, similar to GUIEnableBt. This will enable touch response of the bar.

```
void GUIEnableBar(unsigned short id)

id:        bar ID
```

- **GUIDisableBar**: Disable bar. This will disable touch response of the bar.

```
void GUIDisableBar(unsigned short id)

id:        bar ID
```

## 9.4 API Functions (Sound)

### - Main Sound Functions

Functions in this section are called by system and users don't need to use.

- **SndReadCmd**: Function processes when GUI commands found. Used from [app\\_console.c](#).

```
int SndReadCmd(unsigned int *cons_i)
cons_i:    index of console buffer to be processed
```

return: wrong argument, 0 means no error

- **SndInit**: Initialize objects to default value. Used once before reading script. This is called from GUIInit when sound plug-in is applied.

```
void SndInit(void)
```

- **SndTask**: Task called from [app\\_bs\\_demo.c](#) every 10ms.

```
void SndTask(void)
```

### - Utility Functions

- **SndClrList**: Stop and clear sound list, can be used before SndAddList to play other sound than one that set from script

```
void SndClrList(void)
```

- **SndAddList**: Add sound to list. Using this function alone will wait until playing sound finishes. To play new sound instantly, use SndClrList before SndAddList.

```
void SndAddList(unsigned short snd_no,
unsigned char segment_no)
```

snd\_no: sound ID

segment\_no: segment number of sound (use 0 for non-segmented sound)

- **SndSetOffsetVol**: Before a sound played, the sum of offset volume and individual volume (set from script) will be used to set the chipset.

```
void SndSetOffsetVol(char new_offset_vol)
```

- **SndOn**: Sound is turned on by default; this function turns sound on if SndOff is used.

```
void SndOn(void)
```

- **SndOff**: Turn off sound. Using audio commands cannot stop playing file from sound plug-in. This function will stop current playing and prohibit list addition on the begin of screens.

```
void SndOff(void)
```

- **SndGetCurrentSnd**: Get playing sound ID. In addition, use **VSIIsPlaying()**, it returns 1 if a sound is being played.

```
unsigned short SndGetCurrentSnd(void)
```

## 9.5 External Variables

Some variables, beside of object instances, are declared as external (extern) at the bottom of **gui\_engine.h**. These can be used by users directly.

- **extern unsigned short scr\_id**: Current screen ID, this should only be read by users. To change screen, use **GUIGotoScreen** as described earlier.
- **extern unsigned short old\_scr\_id**: Screen ID, prior to screen change.
- **extern unsigned char scr\_path**: Current path, can be read and written directly.
- **extern unsigned short bt\_snd\_normal**: Sound ID played when normal buttons are pressed.
- **extern unsigned short bt\_snd\_error**: Sound ID played when disabled buttons are pressed.

## 9.6 Programming Guidelines

### - AppScrInit

In [app\\_scr\\_func.c](#), this is the place to initialize all screen init functions (.init). Button's or other object's event functions maybe bound here or in .init of the screen they belong to.

Example code below shows how to bind .init functions. Assume that user\_scr\_initx functions are created by user.

```
void AppScrInit(void)
{
    guisc[0].init = user_scr_init0;
    guisc[1].init = user_scr_init1;
}
```

### - Binding event functions

In screen init functions (.init and .init2), scr\_id can be used to index the screen. Anyway, this cannot be applied to **AppScrInit()**.

Example code below shows how to bind .release of the 3rd button of the first screen to user\_bt\_release function. For the first example, this is done in **AppScrInit()**.

```
void AppScrInit(void)
{
    ...

    guibt[GUIGetBtID(0,2)].release = user_bt_release;
    //0 = first screen
    //2 = 3rd button
}
```

The second example, biding is done in .init function.

```
void user_scr_init0(void)
{
    guibt[GUIGetBtID(scr_id,2)].release = user_bt_release.
    //this affects the same as
    //guibt[GUIGetBtID(0,2)].release = user_bt_release.
    //since scr_id is 0 here
```



}

## - init, init2 and snd\_end

Care should be taken on doing something in .init and .init2 functions. Due to the fact that .init is called before everything is drawn on the screen. And when finish drawing, .init2 is called. So it's very important to know where your code should locate on or you will lose an hour for debugging. There are 3 rules to remember.

1. To enable, disable or change some parameters on objects, do it in .init so the GUI engine use updated parameter to draw out objects. Here you won't need to call those object-drawing functions such as **GUIRefreshBt**, **GUIRefreshTxt** or **GUIShowImg**.
2. To add text to a textbox, change sound in some special case, reset key textbox, do it in .init2. Doing anything later on the screen has the same condition with .init2. Here you need object-drawing functions after enabling or disabling some objects.
3. Whatever does nothing with the screen or objects can be called either in .init or .init2.

snd\_end is an event added later for screens that need sound playing finished before other operations begin. Every time a sound file is added to the list, a flag set and the when the list ended, snd\_end is called. But as sound can be turned off in some applications, snd\_end still be called once. Even that sound plug-in is off; snd\_end will be called just after the init2.

## - Sharing event functions

In some applications, many buttons can have similar function. The difference might be number, quantity, or string. Consider the application of automatic popcorn vending machine. The machine can sell small, medium and big size of popcorn. And their prices are 1\$, 3\$ and 5\$ respectively.

Size	small	medium	big
Price(\$)	1	3	5

On the screen, we have 3 pictures of popcorn with their size. These 3 pictures can be pressed as buttons. And whenever one of them is pressed, the same operation

begins. The screen shows total money the customer has to pay for popcorn. And wait for coins to be inserted.

To bind these buttons to the same function, they need a common parameter. Here we will set their 'cmd' with "BT X 'p'", we use 'p' for 'Popcorn'. And then we set their price to button's value, for example we set "BT V 1" for the small size.

Scripts below show what would come out.

```
//small popcorn
GUI BT X 'p'
GUI BT M ...
GUI BT S
GUI BT V 1
GUI BT N ...
GUI BT P ...
GUI BT E

//medium popcorn
GUI BT L
GUI BT V 3
GUI BT N ...
GUI BT P ...
GUI BT E

//big popcorn
GUI BT L
GUI BT V 5
GUI BT N ...
GUI BT P ...
GUI BT E
```

In the code, we first have to look for popcorn buttons by their 'cmd'. This could be done in **AppScrInit()** or the init function for the related screen. And we bind their release function with **popcorn\_bt\_release**.

```
void AppScrInit(void)
{
    int i;
    ...

    for (i=0;i<MAX_BUTTON;i++)
    {
        if (guibt[i].cmd == 'p')
            guibt[i].release = popcorn_bt_release;
    }
}
```

Then in **popcorn\_bt\_release**, we read its value and set to a global variable ‘price’ for next process.

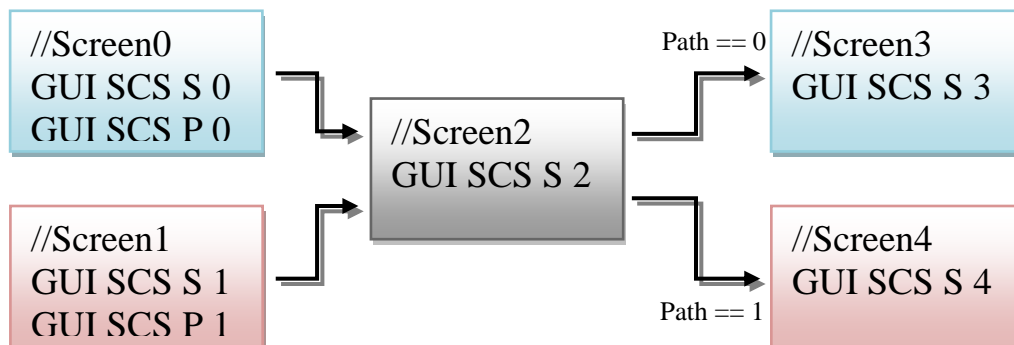
```
void popcorn_bt_release(void)
{
    unsigned short cid = ScrGetCurrentObj();
    price = guibt[bt_id(scr_id,cid)].aux_val;
}
```

Notice the **ScrGetCurrentObj** function, it returns the current object ID on the current screen. The returned value reflects the button order on the current screen, but not for other type of object.

In the same way with popcorn price, if one needs to printout what’s bought on the bill, button name can be set. For example, “BT n Small popcorn (16 oz.)” can be set to the first button.

## - Using path

Using path could be helpful in GUI design when there are some screens that are shared among 2 or more routs. For example, see figure below.



Notice the lines “GUI SCS P x” from the scripts in screen 0 and 1. These set path value whenever the screen 0 or 1 begins. From the screen 2, the user can determine next screen using the value of path (it’s ‘scr\_path’ in the code).

Further more, having a “Next” button on screen 2 with “BT G 3 4”, the GUI Engine will automatically determine next screen looking from the route without any code needed.

## 10. Extension Script

Extension script is the script that runs when events occur or specific screen shown, unlike the main script (main.txt) that is read once before the GUI start. Extension script is not designed for GUI but for simple actions activated by GUI components. At the time of the latest GUI Script example released, extension script can be activated by buttons and screens.

### 10.1 Commands in Extension Script

There can be any console command in extension script. For convenience, we have created 4 more commands in the example project to support extension script.

*-IO port 0/1<enter>*

Drive IO port to 0 or 1, the port number can be 0-23. These ports are in GPIO connector P8 and P12.

*-DELAY n<enter>*

Delay n time of 100 ms.

*-BAUDRATE1 rate <enter>*

Set baud rate of serial port 1. The rate value can be, for example, 115200, 57600, 9600. This command might be set only once and can be placed in the main script file.

*-SER1 "text"<enter>*

Print out text from serial port 1.

### 10.2 Attach Script to Components

A button or a screen can be set to read one extension script file by GUI command "BT s" and "SCS s" respectively (see chapter 5 for more detail). But to support

## 10.3 Events

There are more than one events can be activated by a button or a screen, to support multiple event in a single script file, the start and stop text must be used to separate between events.

The start text must be the event name, for buttons, events supported are <PRESS> and <RELEASE>. For screens, events supported are <INIT>, <INIT2> and <SND\_END>. And the stop text is <END>.

To illustrate how to use extension script file, we use the sun7.txt in “sr” folder from the example project as an example (shown on the right).

The script file is attached to the SUN7 button in the 4<sup>th</sup> screen with a “BT s” command.

```
GUI BT M 2 2
GUI BT S
GUI BT O 7 160
GUI BT N 4
GUI BT P 5
GUI BT n SUN7
GUI BT s sun7.txt
GUI BT E
```

When the button is pressed, the GUI engine searches for the text “<PRESS>” in “sun7.txt”, when found it read and execute the command line by line until the text “<END>” found. The result is the IO0-IO7 will be driven high for 200 ms respectively and a text “SUN7” will be printed out from serial port 1 (the baud rate is set to 9600 at the end of “main.txt”).

```
<PRESS>
IO 0 1
DELAY 2
IO 0 0
DELAY 2
IO 1 1
DELAY 2
IO 1 0
DELAY 2
IO 2 1
DELAY 2
IO 2 0
DELAY 2
IO 3 1
DELAY 2
IO 3 0
DELAY 2
IO 4 1
DELAY 2
IO 4 0
DELAY 2
IO 5 1
DELAY 2
IO 5 0
DELAY 2
IO 6 1
DELAY 2
IO 6 0
DELAY 2
IO 7 1
DELAY 2
IO 7 0
DELAY 2
SER1 SUN7
<END>
```

## 11. SUN7 Studio

SUN7 Studio is free graphic-based software developed by ThaiEasyElec.com. It's used to develop GUI for SUN7. The output of the software is a script folder that the user can take it to run the board. No script has to be written or learned anymore.

### 11.1 To get SUN7 Studio

SUN7 Studio is a freeware downloadable from our website using this link:

<http://www.thaieasyelec.net/archives/Installer/SUN7%20Studio%20Setup.exe>

Anyway, to generate output, you will need a unique key to activate. Please use computer ID code (CID) to fill in our registration form here:

<https://docs.google.com/a/thaieasyelec.com/spreadsheet/viewform?formkey=dEhwLXJHdFdySzJlWl95SFZmdEo0OHc6MQ>

Key will be sent back via email, use it to activate the software.

## 11.2 User Interface Description

- Overall

**Menu bar**

**Screen**  
Design your screens here

**List**  
Objects on current screen are listed here in order with their names

**Property**  
Selected object's properties are shown here

**Position**  
Selected object's position on the screen

**Function**  
Set background or add new object with these functions

**Screen Settings**  
Settings for selected screen

The screenshot shows the SUN7 STUDIO V1.1 software interface. The main workspace is titled 'Home' and contains several UI elements: a 'Language' button with a flag icon, a 'Button' with a play icon, a 'Textbox' with the number '1234', an 'Image Font' button with '1234,' text, a 'Table' with a grid, a 'Percent Bar' with a bar chart, a 'Popup' with a screen icon, and a 'Free Style' button. The left toolbar includes options for Background, Image, Image Button, Color Button, TextBox, Label, Box, Table, Bar, and Pop Up. The right side features a 'List' panel showing a hierarchy of objects (SCS0, Background, home, language, button, textbox, image\_font, table, bar, popup, free\_style) and a 'Property' panel showing the 'Background color' as 255,255,255. The bottom panel contains 'Screen Settings' for the selected screen (SCS0), including 'Show Time' (0 Sec), 'Next screen' (SCS0), and 'General Purpose Variables' (Char Type, Long Type, Keypad, Sound).

## - Menu bar

### *File*

- |                       |                           |
|-----------------------|---------------------------|
| - <i>New project</i>  | Create new project        |
| - <i>Open project</i> | Open an existing project  |
| - <i>Save As</i>      | Save project as a new one |
| - <i>Save</i>         | Save current project      |

### *Edit*

- |                 |                        |
|-----------------|------------------------|
| - <i>Copy</i>   | Copy selected object   |
| - <i>Paste</i>  | Paste copied object    |
| - <i>Delete</i> | Delete selected object |

### *Tool*

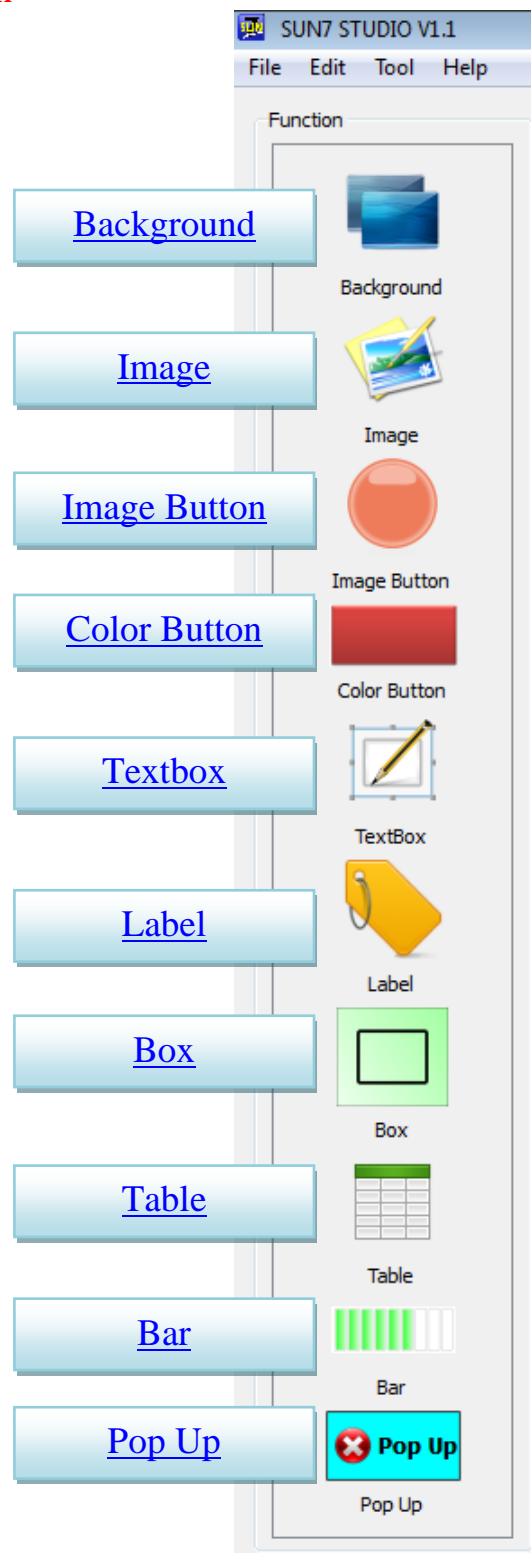
- |   |                        |
|---|------------------------|
| - <i>New screen</i>                       | Create new screen      |
| - <i>Generate</i>                         | Generate script folder |
| - <i><a href="#">Project settings</a></i> | Click for more detail  |

### *Help*

- |                            |                      |
|----------------------------|----------------------|
| - <i>About SUN7 Studio</i> | Software information |
|----------------------------|----------------------|



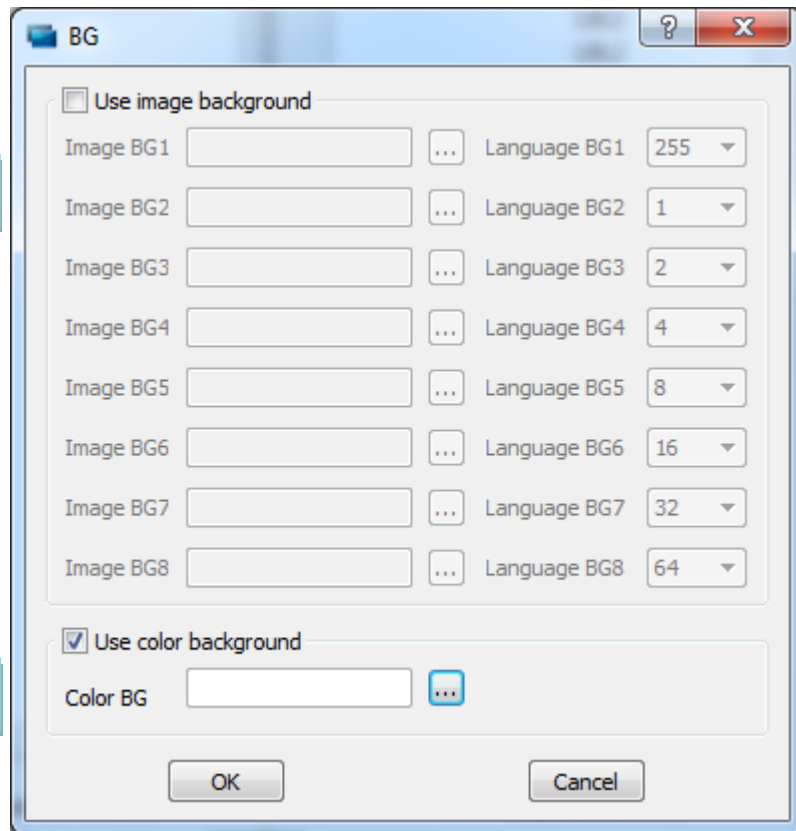
## - Function



## Function: Background

[RBG, BG A](#)

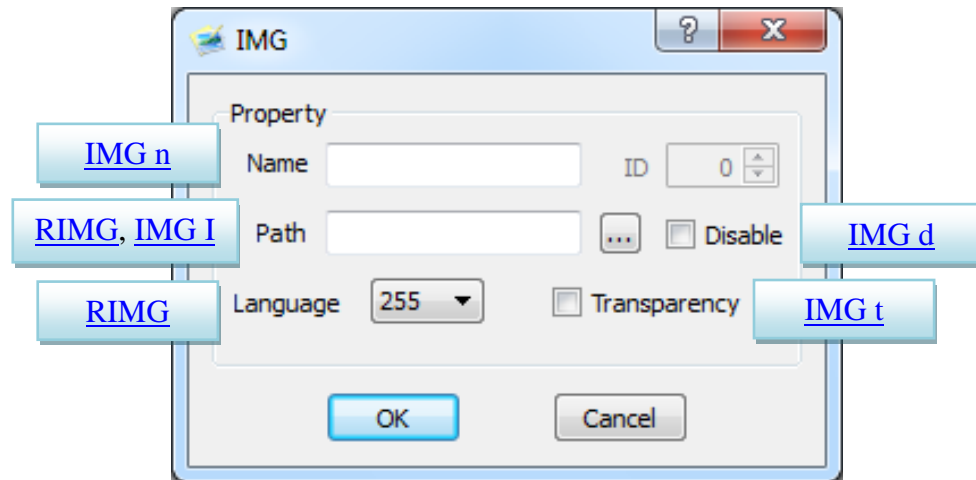
[BG C](#)



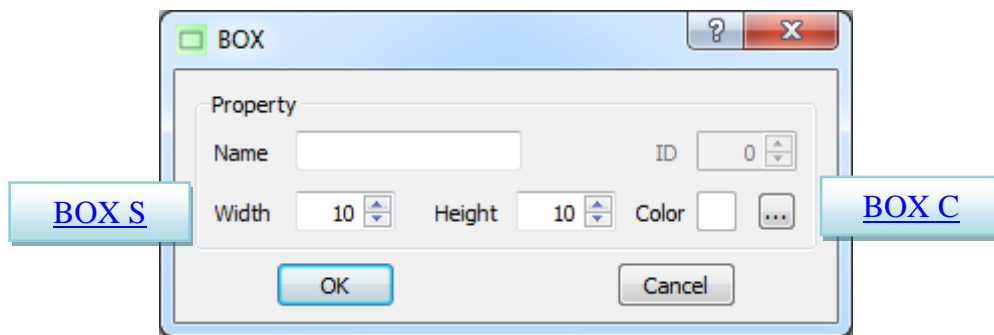
The image shows a dialog box titled "BG" with a standard Windows-style title bar (minimize, maximize, close buttons). The dialog is divided into two main sections. The first section, "Use image background", is currently unchecked. It contains eight rows, each with a label "Image BG1" through "Image BG8", a text input field, a browse button (three dots), and a "Language BG" dropdown menu. The dropdown values are 255, 1, 2, 4, 8, 16, 32, and 64 respectively. The second section, "Use color background", is checked. It contains a "Color BG" label, a white text input field, and a color selection button (a small square with a color gradient). At the bottom of the dialog are "OK" and "Cancel" buttons.

Image BG	Language BG
Image BG1	255
Image BG2	1
Image BG3	2
Image BG4	4
Image BG5	8
Image BG6	16
Image BG7	32
Image BG8	64

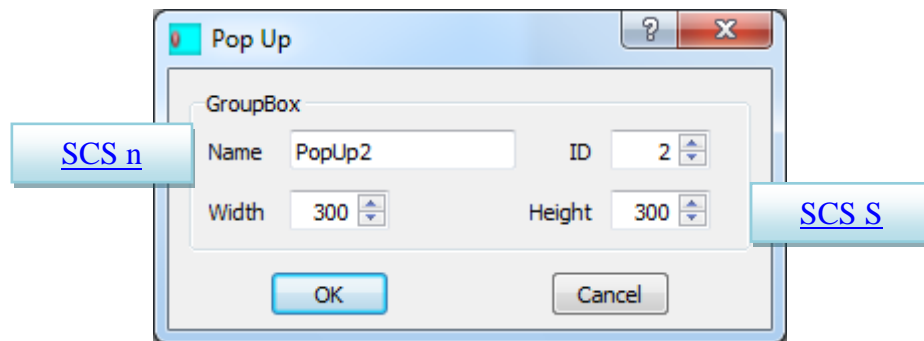
## Function: Image



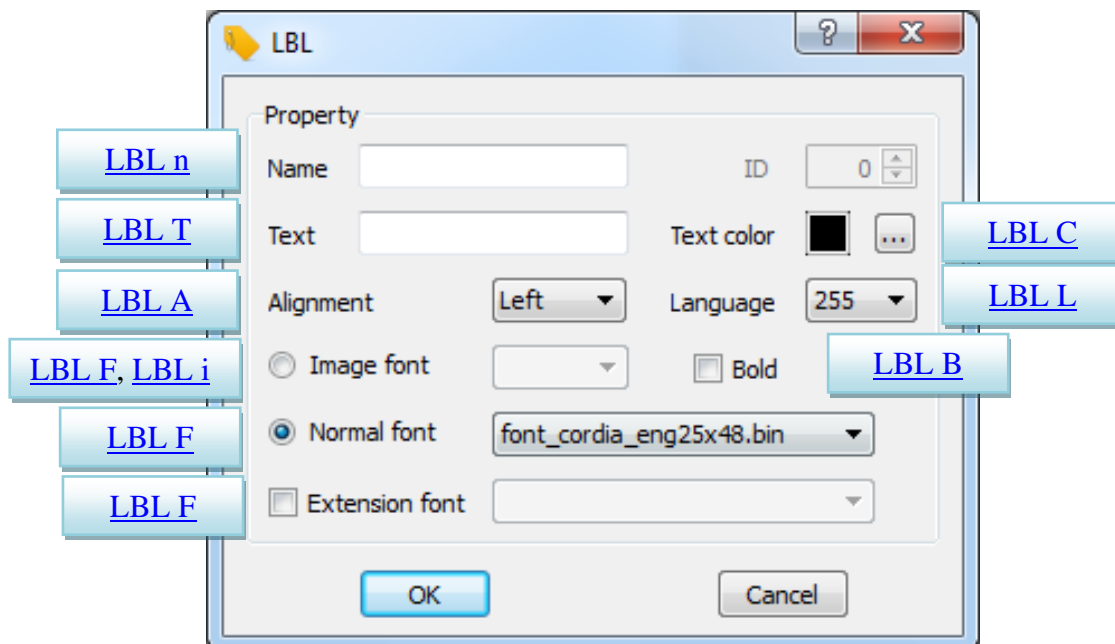
## Function: Box



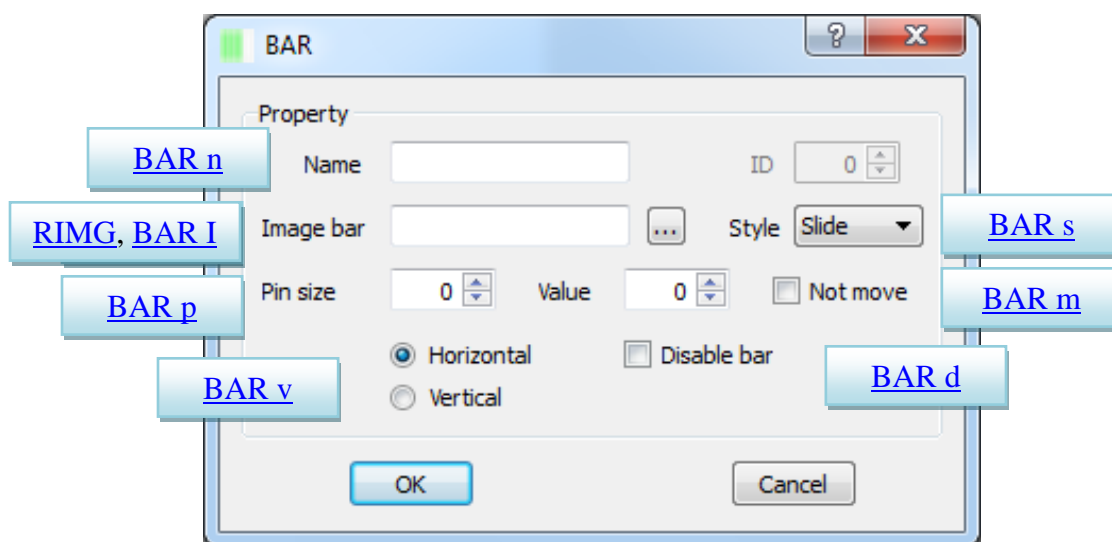
## Function: Pop Up



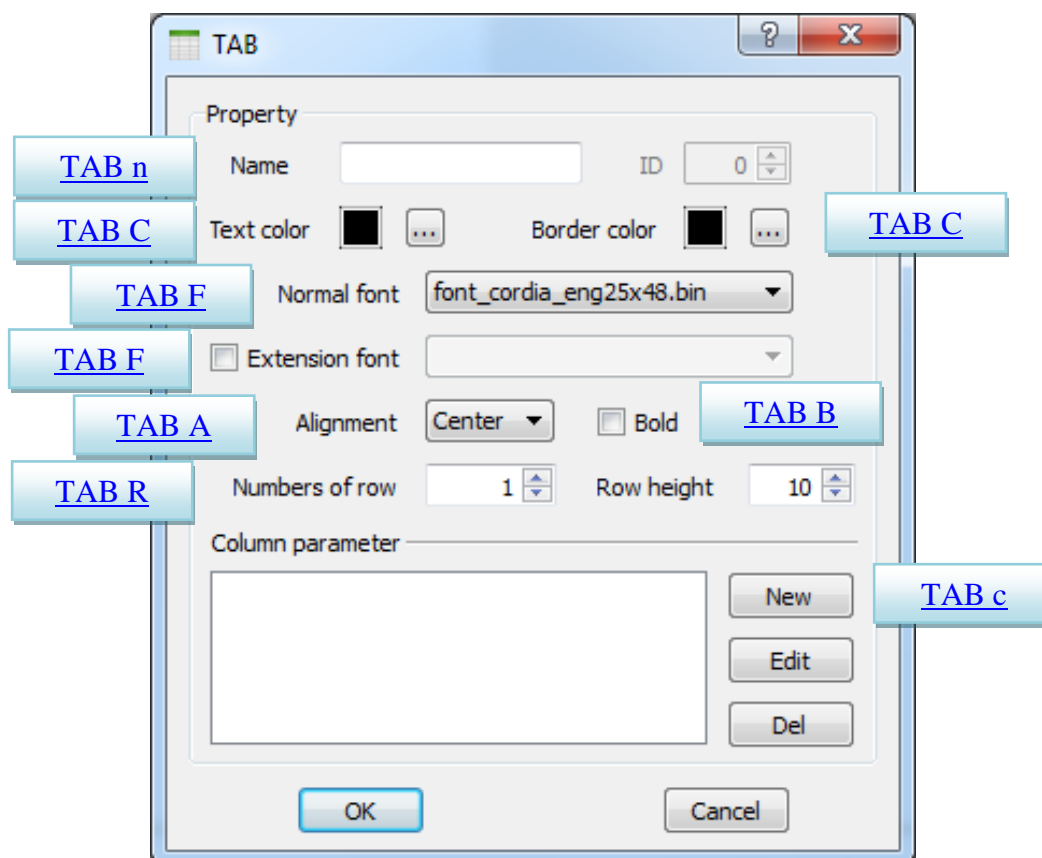
## Function: Label



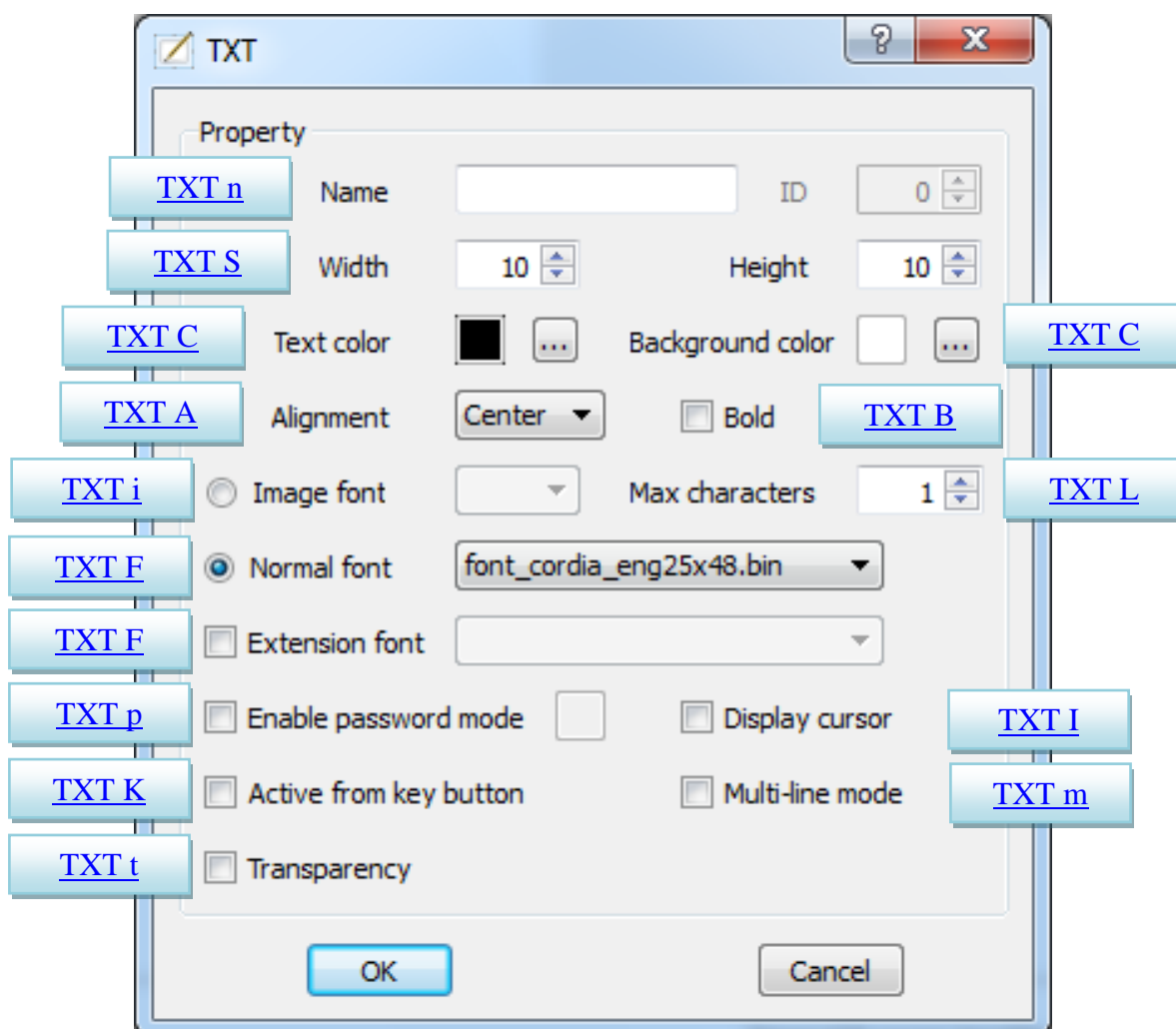
## Function: Bar



## Function: Table



## Function: Textbox



## Function: Image Button

The image shows a dialog box titled "BT1" with a "Property" tab. The dialog contains various settings for an image button, including Name, ID, Language, image files (normal, press, disable), text color, font, and various action options like "Goto", "Repeat", "Transparency", etc. The dialog is surrounded by a series of blue navigation buttons labeled with underlined text: BT n, RIMG, BT N, RIMG, BT P, RIMG, BT D, BT C, BT F, BT F, BT G, BT V, BT r, BT t, BT i, BT s, RIMG, BT M, BT A, BT B, BT X, BT I, BT p, BT c, and BT d.

**BT1**

Property

Name  ID  Language

Img normal  ...  Move X

Img press  ...  Move Y

Img disable  ...  Alignment

Text color  ...  ☐ Bold

Normal font

☐ Extension font

☒ Not goto ☐ Goto next screen ☐ Goto screen

☐ Goto path

☐ Value  ☐ Special value

☐ Repeat button ☐ Interlock ID

☐ Transparency ☐ Open popup window

☐ Invert mode ☐ Close popup window

☐ Extension script  ☐ Disable button

BT n RIMG, BT N RIMG, BT P RIMG, BT D BT C BT F BT F BT G BT V BT r BT t BT i BT s RIMG BT M BT A BT B BT X BT I BT p BT c BT d

## Function: Color Button

**BT2**

Property

Name  ID  Language

Color normal ☐ ...  Width  Height

Color press ☐ ...  Move X  Move Y

Color disable ☐ ...  Alignment

Text color ☐ ...  Text  ☐ Bold

Normal font

☐ Extension font

☒ Not goto ☐ Goto next screen ☐ Goto screen

☐ Goto path

☐ Value  ☐ Special value

☐ Repeat button ☐ Interlock ID

☐ Transparency ☐ Open popup window

☐ Invert mode ☐ Close popup window

☐ Extension script  ☐ Disable button

BT n

RIMG, BT N

RIMG, BT P

RIMG, BT D

BT C

BT F

BT F

BT G

BT V

BT r

BT t

BT i

BT s

RIMG

BT M

BT A

BT B

BT X

BT I

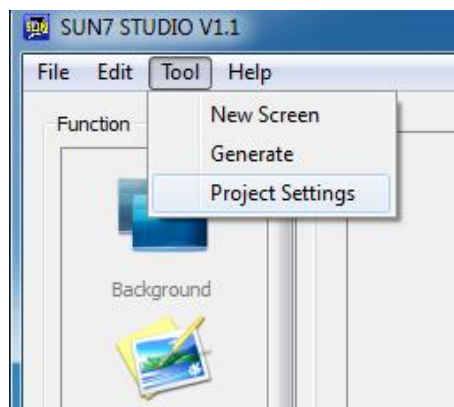
BT p

BT c

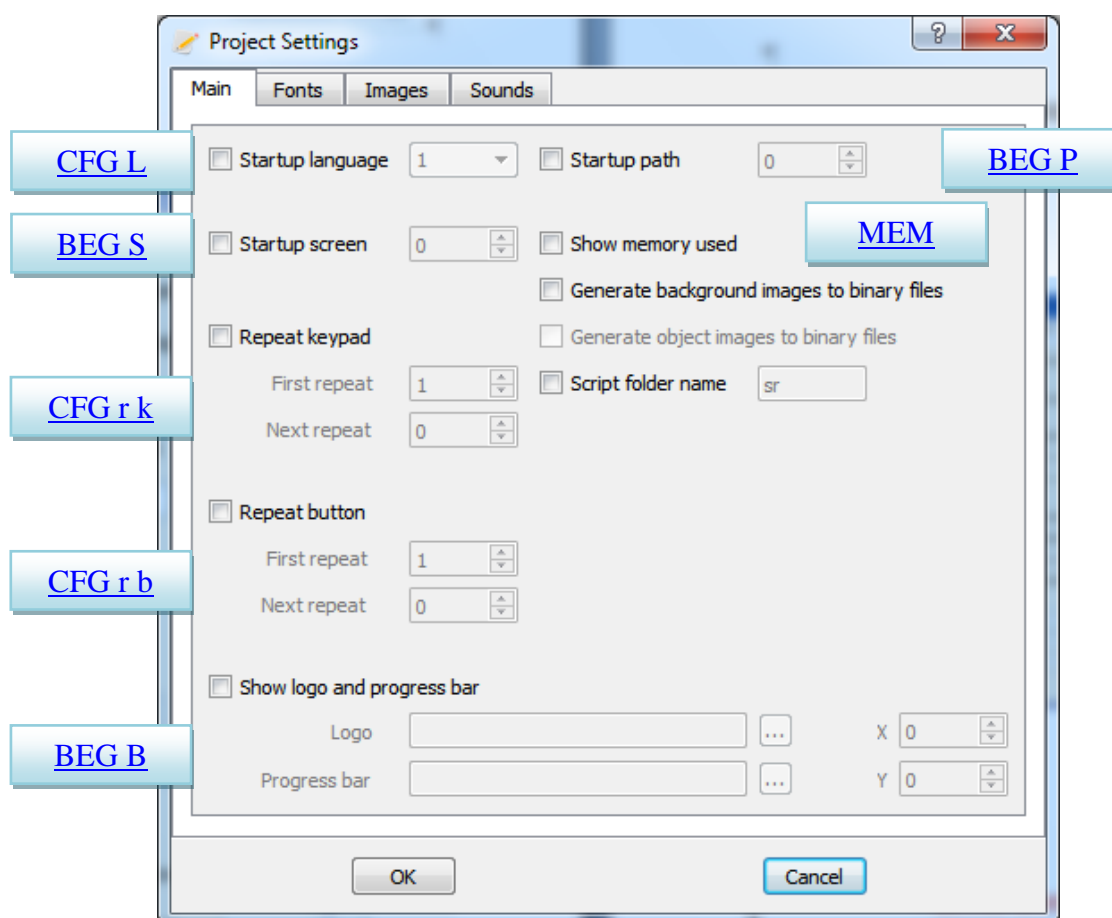
BT d



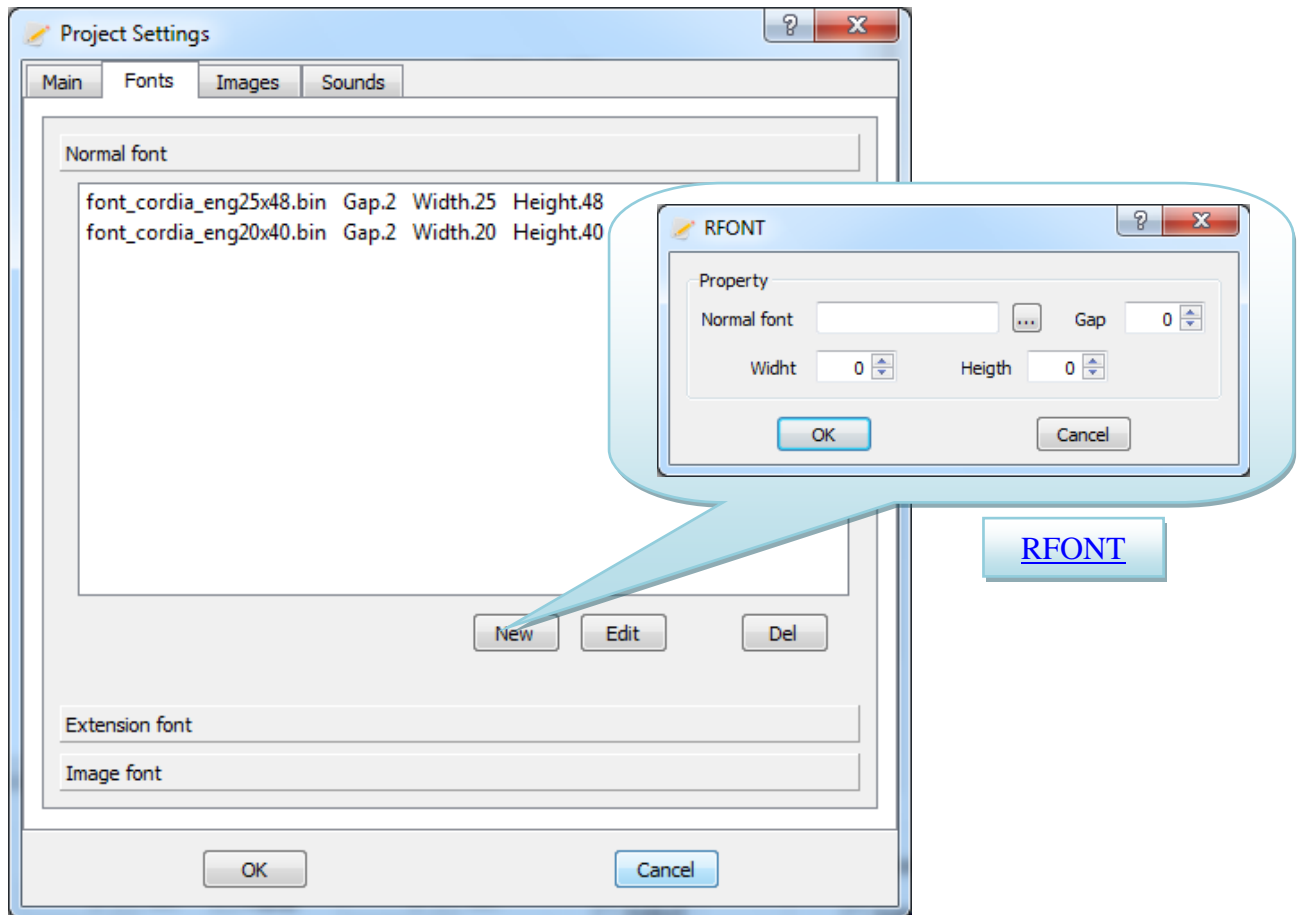
## - Project Settings



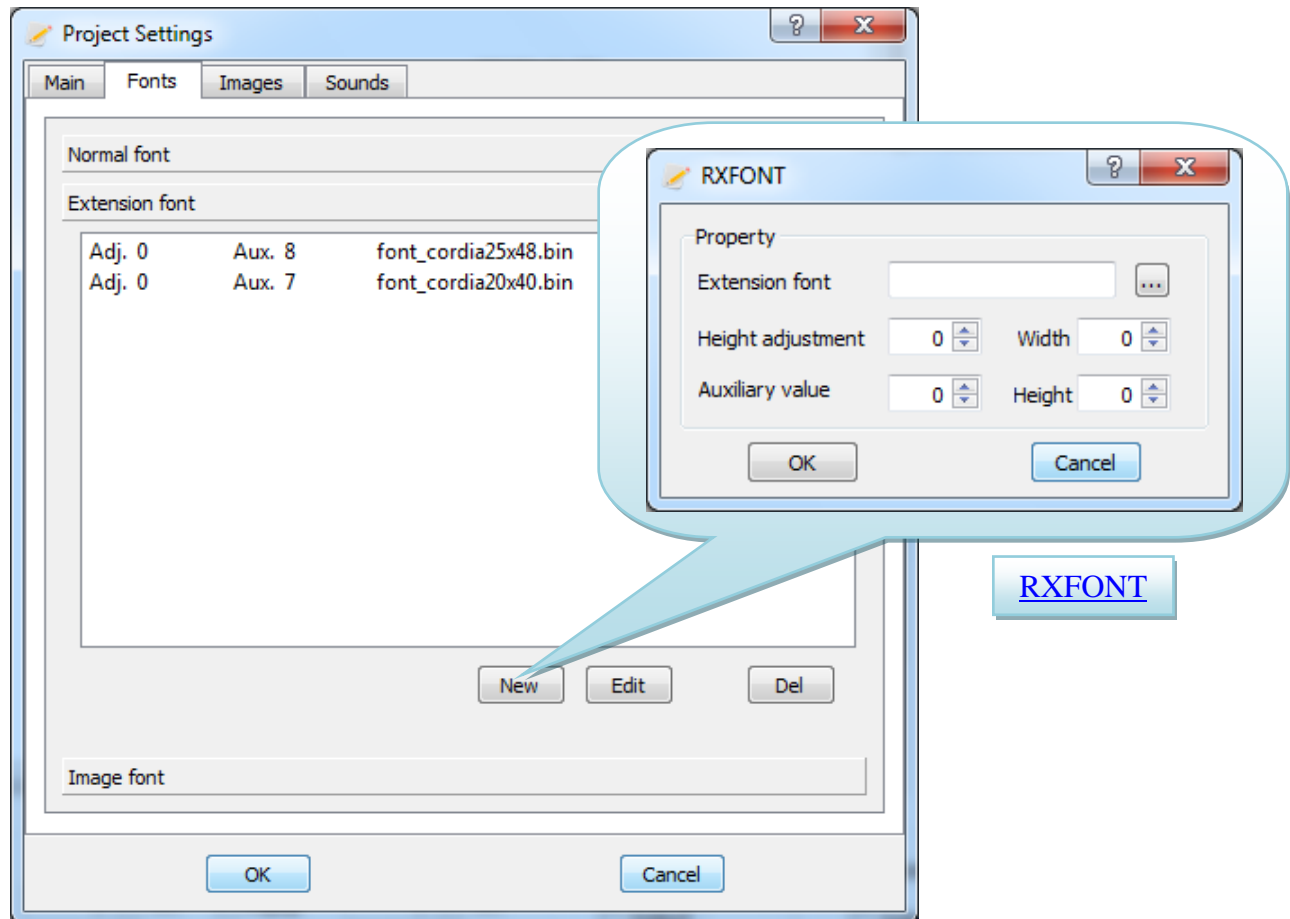
### Project Settings: Main



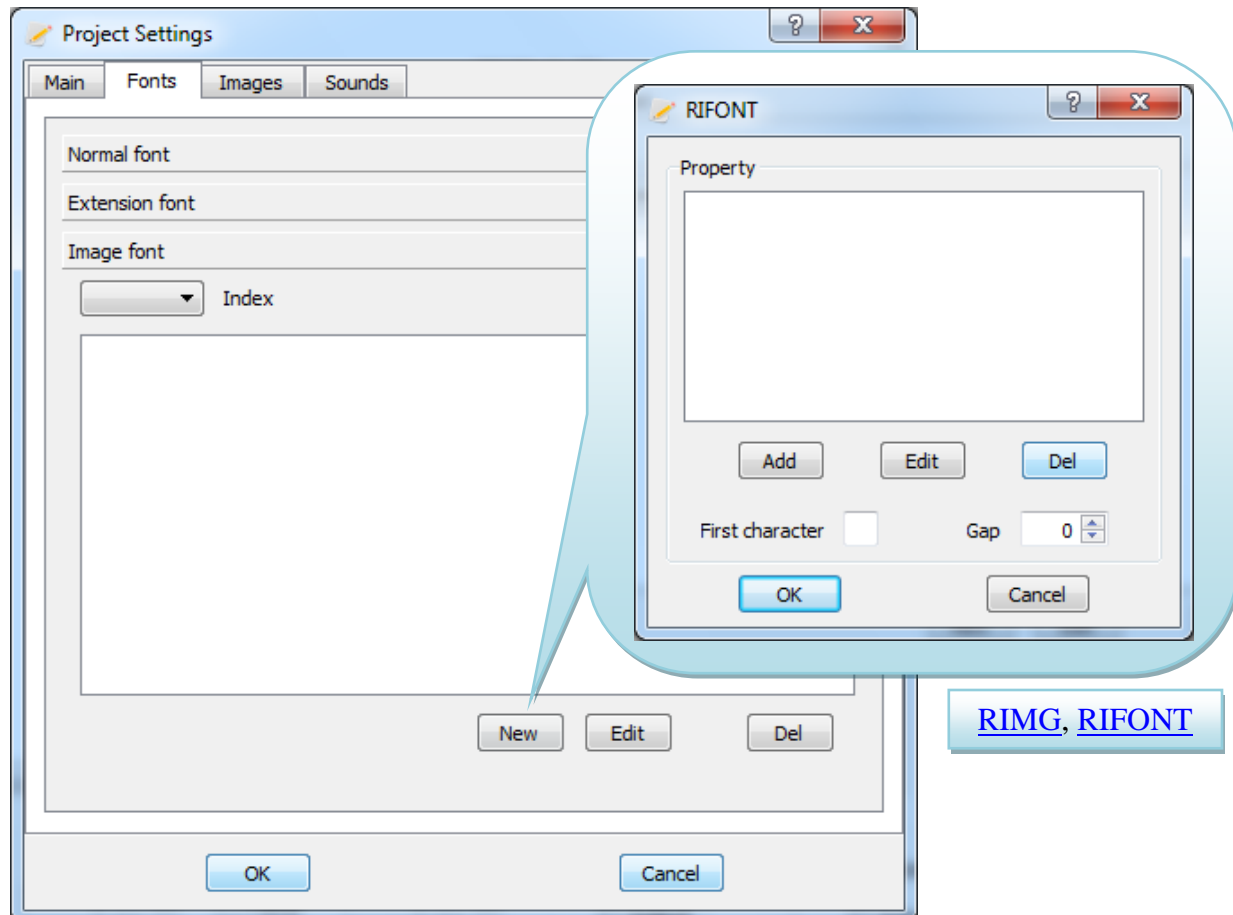
## Project Settings: Normal font



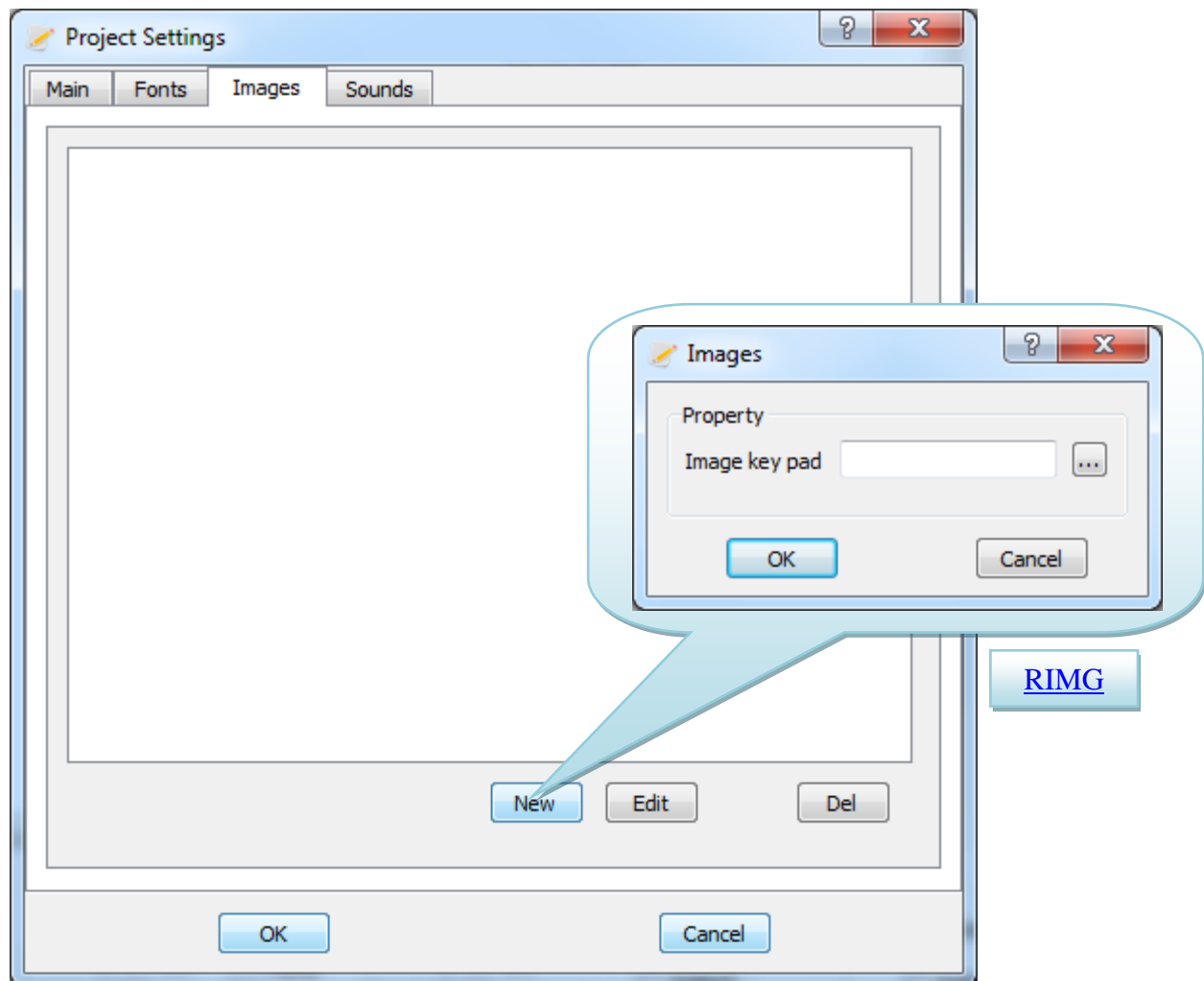
## Project Settings: Extension font



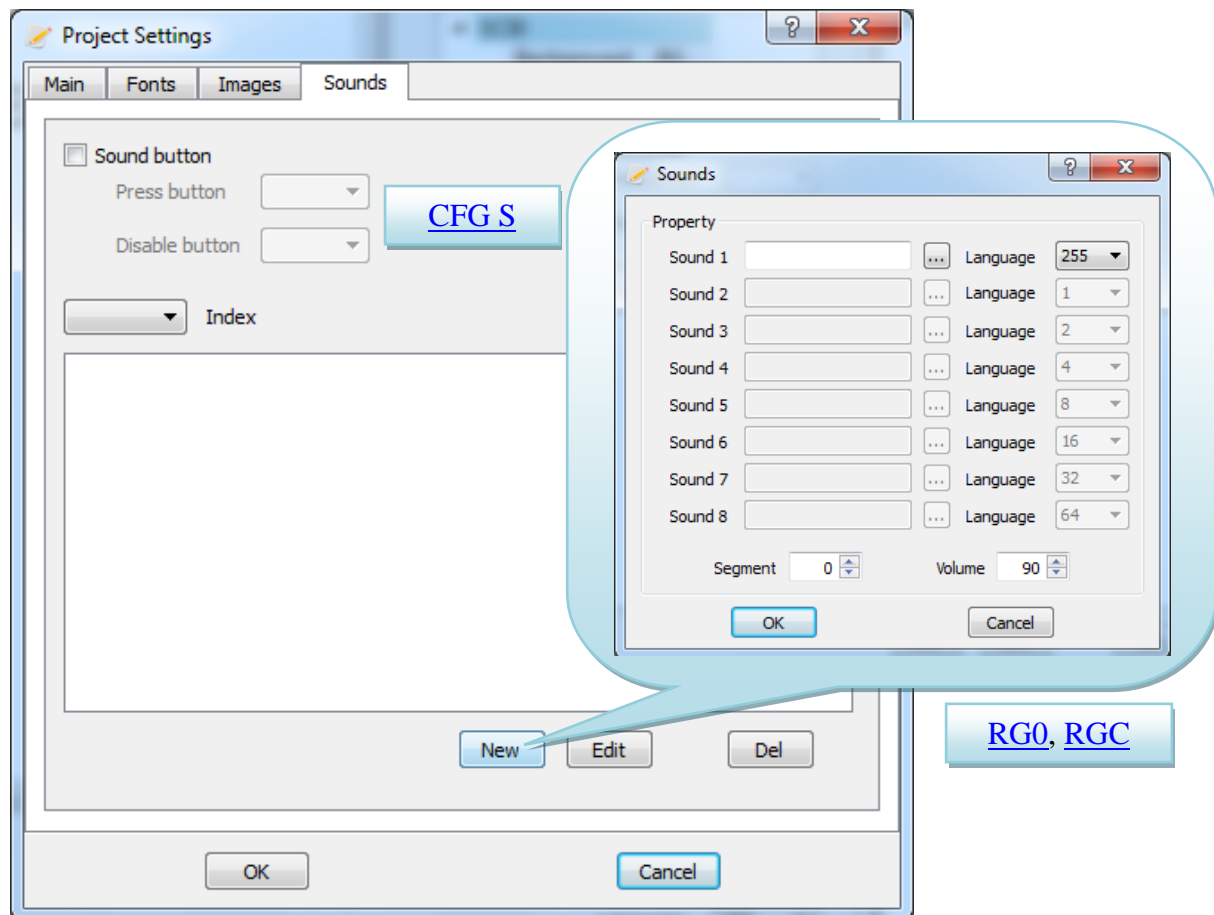
## Project Settings: Image font



## Project Settings: Images



## Project Settings: Sounds



## - Screen Settings

[Show time](#)

[General purpose variables](#)

**Name and ID**

Name: SCS0

ID: 0

Del SCR Edit

☐ Path

**Show Time**

☐ Screen 0 Sec ☐ Popup 0 Sec

☒ Next screen ☐ Custom screen SCS0

☐ Goto path

**General Purpose Variables**

Char Type

☐ Index0  

☐ Index1  

☐ Index2  

☐ Index3

**Long Type**

☐ Index0  

☐ Index1

**Keypad and Sound**

Language: 255

☐ Keypad Default

☐ Sound

[Name and ID](#)

[Keypad and sound](#)

### Screen Settings: Name and ID

**Name and ID**

Name: SCS0

ID: 0

Del SCR Edit

☐ Path

Delete screen

[SCS P](#)

[SCS S](#)

## Screen Settings: Show Time

SCS W

SCS W

SCS c

## Screen Settings: General purpose

SCS C

SCS L

## Screen Settings: Key and sound

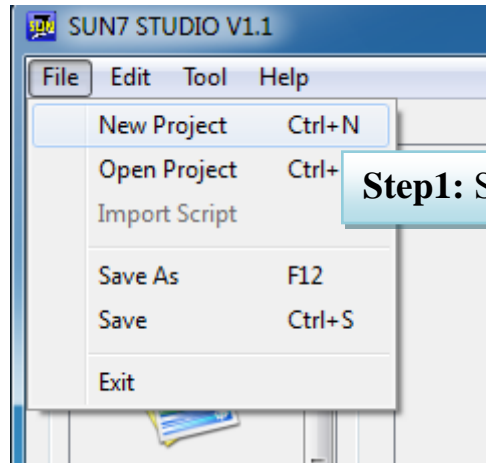
SCS K

SND

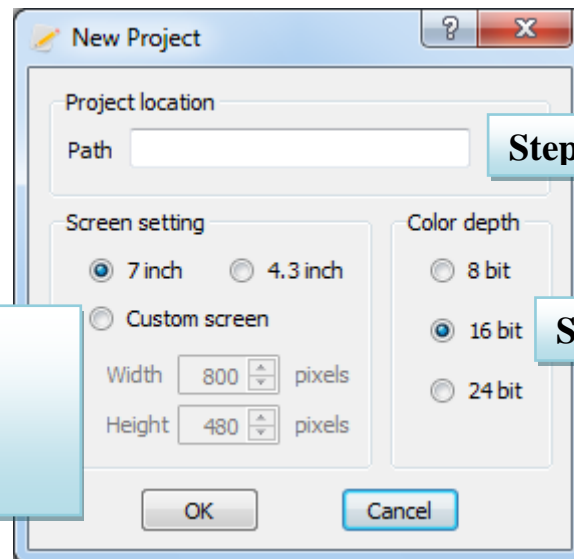
Select language for display



## 11.3 To create new project



**Step1:** Select “New Project”



**Step2:** Choose path to save

**Step3:** Choose screen size  
7inch, 4.3inch  
or custom size.

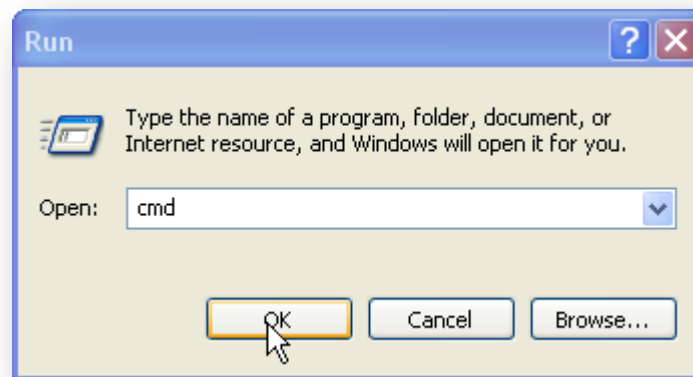
**Step4:** Choose color depth

## Appendix

### 1. Installation of Eclipse and Yagarto

#### 1.1 Install Java Runtime Environment (JRE)

Select “Start Menu” from your Windows and then “Run”, type “cmd” and hit enter.



Type “java –version” to see Java Runtime version on your machine.

```
C:\> java -version
java version "1.6.0_13"
Java(TM) SE Runtime Environment (build 1.6.0_13-b03)
Java HotSpot(TM) Client VM (build 11.3-b02, mixed mode, sharing)
C:\> _
```

If this is not installed on your machine, download it from here:

<http://java.sun.com/javase/downloads/index.jsp> .

The version using in this application note is Java Runtime Environment (JRE) 6 Update 13

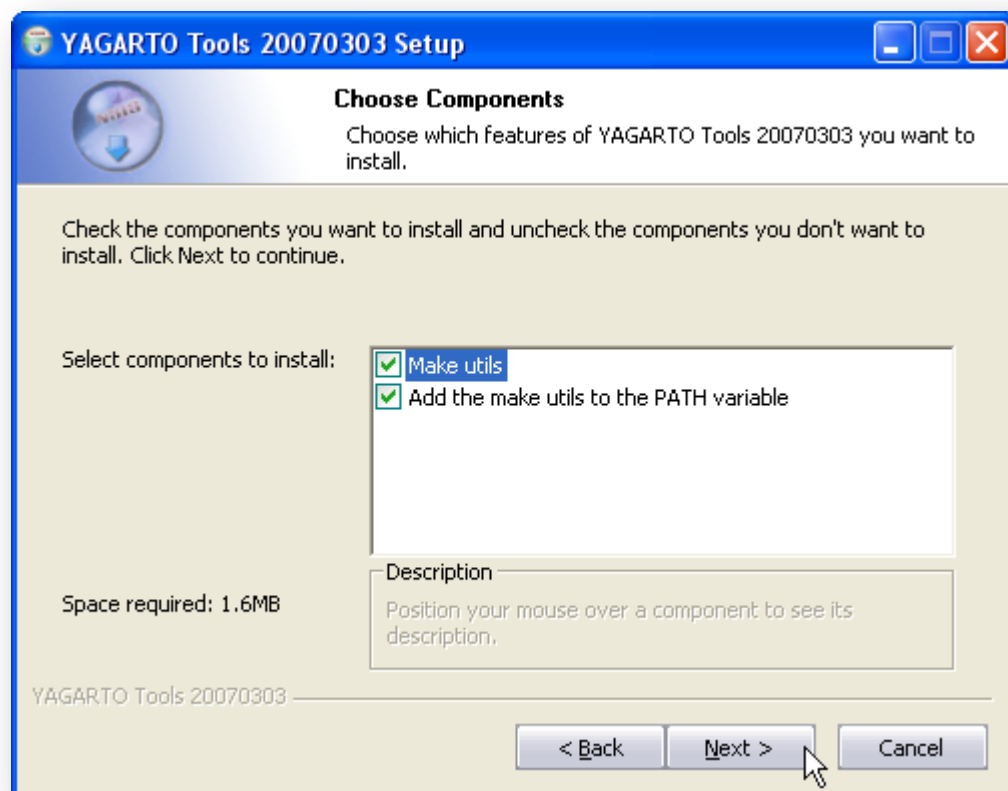
## 1.2 Install Eclipse IDE for C/C++ Developers

Eclipse IDE for C/C++ Developers can be downloaded from:

<http://www.eclipse.org/downloads/> . There you will get file eclipse-cpp-ganymede-SR2-win32.zip. Unzip it and install on your machine.

## 1.3 Install Yagarto Tools and Yagarto GNU ARM Toolchain

Download from <http://www.yagarto.de/index.html> . Follow the installation until “Choose Components” page, select as in the picture below.



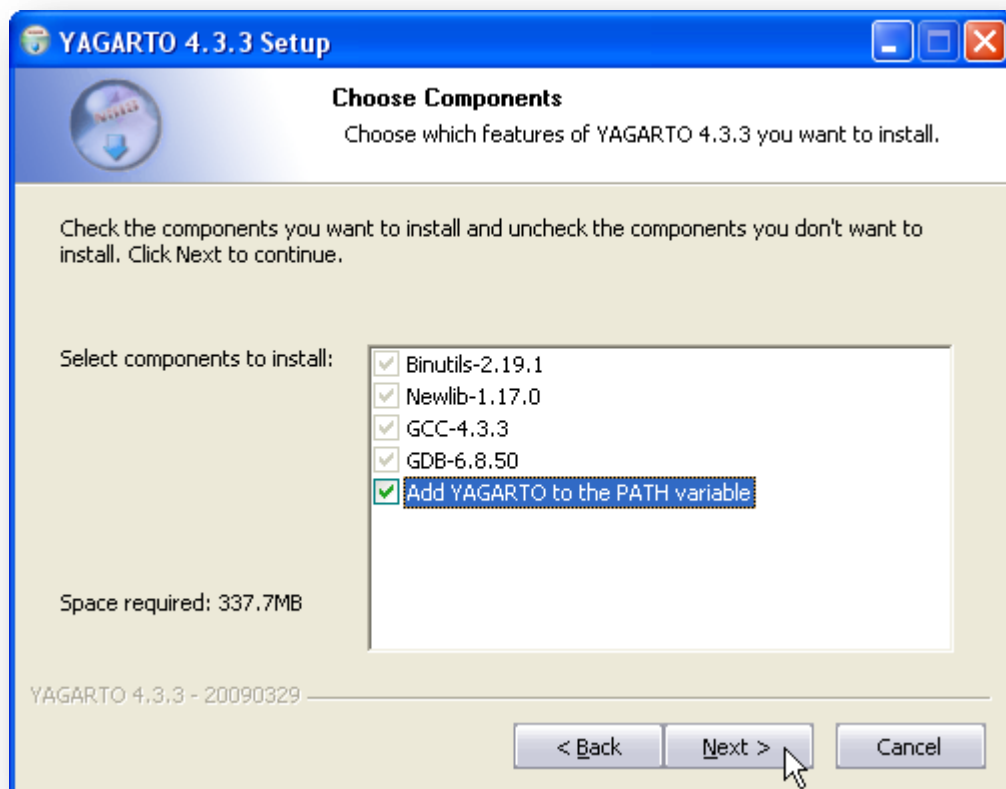
Press “Next” until you finish the installation. Now try on Windows’s command “make --version”, the result should be the same here (unless, the Yagarto Tool may not be installed completely).

```
C:\>C:\WINDOWS\system32\cmd.exe

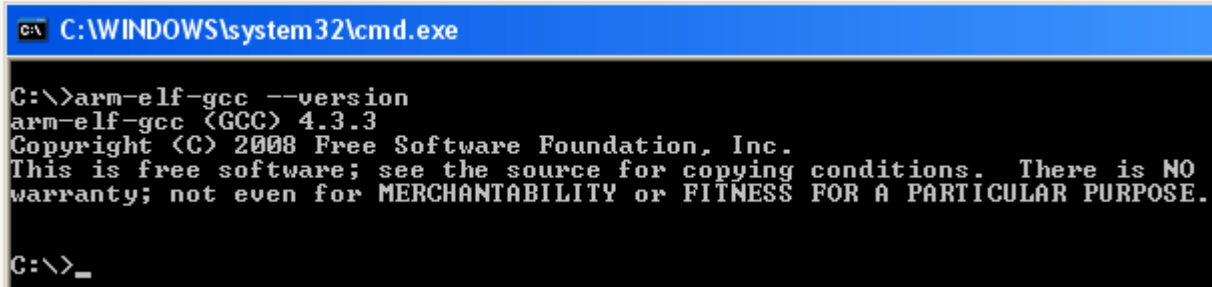
C:\>make --version
GNU Make 3.81
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

This program built for i686-pc-mingw32
C:\>
```

And then install YAGARTO GNU ARM Toolchain until you've got to "Choose Components" page, select as in the picture below.



Press “Next” until the installation is finished. Now test it with command: “**arm-elf-gcc --version**” (for newer version of Yagarto, this might be “**arm-none-eabi-gcc --version**”).



```
C:\WINDOWS\system32\cmd.exe

C:\>arm-elf-gcc --version
arm-elf-gcc (GCC) 4.3.3
Copyright (C) 2008 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

C:\>_
```

## 2. Command List

Command	Format	Section
Register background	GUI RBG <i>ID Image_type Lang Path</i>	5.1
Register fonts	GUI RFONT <i>ID Width Height Gap Path</i>	5.2
Register extension fonts	GUI RXFONT <i>ID Width Height Adj Aux Path</i>	5.3
Register image fonts	GUI RIFONT <i>ID 1st_char nChar Image_ID Gap</i>	5.4
Register images	GUI RIMG	5.5
-Binary files	GUI RIMG <i>ID 0 Width Height Lang Path</i>	
-BMP files	GUI RIMG <i>ID 2 Lang Path</i>	
-Color box	GUI RIMG <i>ID 3 Width Height Lang Color [Border]</i>	
Show memory status	GUI MEM	5.6
Screen settings	GUI SCS	5.7
- ID	GUI SCS S <i>ID</i>	
- ID (popup window)	GUI SCS S <i>ID Width Height</i>	
-Name	GUI SCS n <i>Name</i>	
-Switch path	GUI SCS P <i>Path</i>	
-Auxiliary value (char)	GUI SCS C <i>Index Value</i>	
-Auxiliary value (long)	GUI SCS L <i>Index Value</i>	
-Wait time (all paths)	GUI SCS W <i>Time [ID]</i>	
-Wait time (multi-path)	GUI SCS W <i>Time ID#1 ID#2 ...</i>	
-Wait time to close (popup)	GUI SCS c <i>Time</i>	
-Keypad	GUI SCS K <i>Keypad_type</i>	
-Extension script setting	GUI SCS s <i>Filename</i>	
Background settings	GUI BG	5.8
-Add to screen	GUI BG A <i>ID#1 [ID#2] [ID#3] ... [ID#8]</i>	
-Set background color	GUI BG C <i>Color</i>	
Image box settings	GUI IMG	5.9
-Origin	GUI IMG O X Y	
-Image ID	GUI IMG I <i>ID</i>	
-Transparency	GUI IMG t	

Command	Format	Section
-Disabled at startup	GUI IMG d	
-Name	GUI IMG n <i>Name</i>	
-End setting	GUI IMG E	
Button settings	GUI BT	5.10
-Name	GUI BT n <i>Name</i>	
-Origin	GUI BT O X Y	
-Normal state image	GUI BT N ID	
-Pressed state image	GUI BT P ID	
-Disabled state image	GUI BT D ID	
-Shift interlock button	GUI BT I <i>Order</i>	
-Movement	GUI BT M X Y	
-Screen to go (all paths)	GUI BT G [ID]	
-Screen to go (multi-path)	GUI BT G ID#1 ID#2 ...	
-Disabled at startup	GUI BT d	
-Transparency	GUI BT t	
-Inverse	GUI BT i	
-On-button text	GUI BT T <i>Text</i>	
-Fonts	GUI BT F ID#1 [ID#2]	
-Bold characteristic	GUI BT B	
-Text color	GUI BT C <i>Color</i>	
-Text alignment	GUI BT A <i>Alignment_type</i>	
-Save button settings	GUI BT S	
-Load button settings	GUI BT L	
-Value	GUI BT V <i>Value</i>	
-Special action	GUI BT X <i>Value</i>	
-Extension script setting	GUI BT s <i>Filename</i>	
-Repetition	GUI BT r	
-Open popup screen	GUI BT p ID	
-Close popup screen	GUI BT c	
-End setting	GUI BT E	
Textbox settings	GUI TXT	5.11
-Origin	GUI TXT O X Y	
-Size	GUI TXT S <i>Width Height</i>	
-Insertion	GUI TXT I	
-Key characters receiver	GUI TXT K	
-Bold characteristic	GUI TXT B	

Command	Format	Section
-Colors	GUI TXT C <i>Text_color Back_color</i>	
-Fonts	GUI TXT F <i>ID#1 [ID#2]</i>	
-Use image font	GUI TXT i	
-Text alignment	GUI TXT A <i>Alignment_type</i>	
-Text length (max. characters)	GUI TXT L <i>Length</i>	
-Password mode	GUI TXT p <i>Password_character</i>	
-Multi-line mode	GUI TXT m	
-Name	GUI TXT n <i>Name</i>	
-Transparency	GUI TXT t	
-End setting	GUI TXT E	
Label settings	GUI LBL	5.12
-Origin	GUI LBL O X Y	
-Bold characteristic	GUI LBL B	
-Color	GUI LBL C <i>Text_color</i>	
-Text	GUI LBL T <i>Text</i>	
-Fonts	GUI LBL F <i>ID#1 [ID#2]</i>	
-Use image font	GUI LBL i	
-Text alignment	GUI LBL A <i>Alignment_type</i>	
-Language	GUI LBL L <i>Lang</i>	
-Name	GUI LBL n <i>Name</i>	
-End setting	GUI LBL E	
Box settings	GUI BOX	5.13
-Origin	GUI BOX O X Y	
-Size	GUI BOX S <i>Width Height</i>	
-Color	GUI BOX C <i>Color</i>	
-End setting	GUI BOX E	
Table settings	GUI TAB	5.14
-Origin	GUI TAB O X Y	
-Bold characteristic	GUI TAB B	
-Color	GUI TAB C <i>Text_color</i> <i>Border_color</i>	
-Column parameters	GUI TAB c <i>Column Width</i> <i>Cap_color Con_color [Text]</i>	
-Row parameters	GUI TAB R <i>Height nRow</i>	
-Fonts	GUI TAB F <i>ID#1 [ID#2]</i>	
-Text alignment	GUI TAB A <i>Alignment_type</i>	
-Name	GUI TAB n <i>Name</i>	



Command	Format	Section
-End setting	GUI TAB E	
Configure global parameters	GUI CFG	5.15
-Default language	GUI CFG L <i>Lang</i>	
-Button sound	GUI CFG S <i>ID#1 [ID#2]</i>	
-Repeat interval for keypad	GUI CFG r k <i>First_interval</i> <i>Next_interval</i>	
-Repeat interval for buttons	GUI CFG r b <i>First_interval</i> <i>Next_interval</i>	
Add sound on screen startup	GUI SND <i>ID</i>	5.16
End GUI settings	GUI END	5.17
Set startup option	GUI BEG	5.18
-Startup screen	GUI BEG S <i>ID</i>	
-Startup path	GUI BEG P <i>Path</i>	
-Boot logo	GUI BEG B <i>ID#1 ID#2 X Y</i>	
Percent bar settings	GUI BAR	5.19
-Origin	GUI BAR O <i>X Y</i>	
-Image ID	GUI BAR I <i>ID</i>	
-Vertical Setting	GUI BAR v	
-Initial value	GUI BAR V <i>Value</i>	
-Touch response style	GUI BAR s <i>Response_style</i> <i>[Sensitivity]</i>	
-Movement type	GUI BAR m <i>Movement_type</i>	
-Pin size	GUI BAR p <i>Pin_size</i>	
-Disabled at startup	GUI BAR d	
-End setting	GUI BAR E	
Configure global parameters (sound)	SND CFG	6.1
-Sound directory	SND CFG D <i>Path</i>	
Register MP3 file	SND RG0 <i>ID nSegment Volume Lang</i> <i>Path</i>	6.2
Register MP3 file (cont.)	SND RGC <i>Lang Path</i>	6.3

**Remark:** Parameters in [ ] are optional