3D GUI LIBRARY FOR TI CAMERAS

User's Guide

24 January 2005

Thank you for your interest in our GUI library for TI cameras. In this manual you will find out how to use it.

Before going on reading the manual, we kindly ask you to read the following

DISCLAIMER

THIS DOCUMENTATION IS PROVIDED FOR REFERENCE PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS DOCUMENTATION, THIS DOCUMENTATION IS PROVIDED “AS IS” WITHOUT ANY WARRANTY WHATSOEVER AND TO THE MAXIMUM EXTENT PERMITTED, ATTO-SYSTEMS LTD. DISCLAIMS ALL IMPLIED WARRANTIES, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SAME. ATTO-SYSTEMS LTD. SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, DIRECT, INDIRECT, CONSEQUENTIAL OR INCIDENTAL DAMAGES, ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS DOCUMENTATION OR ANY OTHER DOCUMENTATION. NOTWITHSTANDING ANYTHING TO THE CONTRARY, NOTHING CONTAINED IN THIS DOCUMENTATION OR ANY OTHER DOCUMENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM ATTO-SYSTEMS LTD. (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF THIS SOFTWARE.

THE DESCRIBED SOFTWARE IS PROVIDED 'AS IS', WITHOUT ANY WARRANTY EXPRESSED OR IMPLIED. NO GUARANTY IS GIVEN THAT THE SOFTWARE IS SUITABLE FOR ANY GIVEN PURPOSE.

COPYRIGHT

TRADEMARKS

All trademarks and copyrights mentioned within the documentation are respected. They are the property of their respective owners.

CONVENTIONS USED IN THIS MANUAL

***INFORMATION**. This sign marks section in the manual, which is for information only.*

***ATTENTION**. This sign marks section of the manual, which is particularly important for the general understanding of the document. Please, make sure to read this section before proceeding with reading the manual.*

***TIPS & TRICKS**. This sign marks a Tips & Tricks section. Here you can find some practical advises on using the system or get a more detailed explanation of some features. Reading this section may help you in solving a particular problem or give you some ideas but is not vital for understanding the document.*

***PREMISE**. This sign marks a section, which requires you to do something before proceeding with reading the manual. Usually this is a demo program, you have to run or something similar.*

File	*Menu item*
File > Open	*Sub-menu item or dialog control*
"1.1 About"	*Section reference. If the section is within the current manual no manual name is specified. When the section is within external manual the name of the respective manual is also included.*
`Ctrl+E`	*Hot-key combination. The first part of the combination specifies which system key to use. Possible values are: Ctrl, Alt, Shift. The second part specifies the normal key to be used in the combination. The plus sign means that you should press these keys simultaneously.*

CONTENTS

1. Introduction

The 3D GUI package is a set of graphics functions for Vision Components cameras, based on Texas Instruments (TI) processor. TI cameras have video-output hardware, which allows connection of a PC monitor to the camera. The camera displays on the monitor a gray-level live picture, received by the camera sensor. The camera generates a second picture, called an "overlay" picture, which is placed over the live picture. The overlay picture has 64 normal colors and 3 translucent colors.

The package contains high-level GUI functions, which can be used to create graphical user interface (GUI) in your application. There are two groups of GUI elements:

- Active GUI elements, which react to mouse or keyboard commands – window, button, edit, spin, radio-button menu, check-box.
- Passive GUI elements, which just draw but do not process user commands – text, frame, log-box.

Refer to "2.13 High-level GUI functions" for more information.

The package contains also low-level drawing functions, which allow general-purpose drawings inside or outside GUI windows. High-level GUI elements are moved together with windows, low-level drawings are static. If you make low-level drawing on current window and move the window, you will lose your drawings.

Using the package is fairly simple. You work in plain C. Your program must initialize the package by `tg_init()` on entry, specify type of user input commands (mouse or keyboard input), and close the package by `tg_exit()` before exit. GUI elements are created (displayed) by single function calls. Passing of user commands to all GUI elements in current window is performed by one function, executed in a loop.

Look at the code of the "Hallo world" program **HALLO.C**, which displays one window and waits for a user command to close the window:

```
#include "tg.h"
void main()
{
        TG_RET   rc;
        TG_INT16 win_id;
        TG_INT16 gui_id;       /* GUI identifier        */
        TG_EVENT gui_event;    /* GUI event             */

/*.............. Init 3D GUI package */
        rc = tg_init();
        if(rc != RC_OK)
        {
           print("3D GUI init error = %d\n",rc);
           return;
        }

/*.............. Set input mode: mouse commands */
        rc = tg_set_inputmode(TG_INPUT_MODE_MOUSE);
        if(rc != RC_OK) goto done;

/*.............. Open window */
        rc = tg_WinOpen("Hallo world...",50,50,300,100,TG_WIN_CLOSE_BUT,
                        2,&win_id);
        if(rc != RC_OK) goto done;

/*.............. Get event loop */
        for(;;)
        {
           rc = tg_GetEvent(&gui_id,&gui_event);
```

```
           if(rc != RC_OK) goto done;

/*............. "Close" button pressed */
           if(gui_id == win_id && gui_event == TG_WIN_EVENT_CLOSE)
           {
              rc = tg_WinClose();
              break;
           }

        } /* for(;;) : get event loop */

/*............. Close 3D GUI package */
done:
        tg_exit();
        if(rc != RC_OK)
           print("RC = %d\n",rc);

        return;
}
```

Compile and link **HALLO.C** by the batch file **TCL.BAT** (see "3.8 Compiling and linking 3D GUI applications") or use the ready MSF file contained in the distribution archive. Upload your registration key file **TGKEY.MSF** to the camera flash (the **FD:** drive). Upload **HALLO.MSF** to camera flash. Start **attoTerminal** or the **TGTERM** terminal program in full DOS window and press **`ALt+M`** to enable sending of mouse commands to the serial port of the camera. Start the demo program. You will see a mouse cursor displayed on the screen. Move the window by pressed left mouse button on the window's title. Close the window by left mouse click on the window's close button.

You can modify the program by commenting the function `tg_set_inputmode()`. Now you will execute the program in keyboard input mode (default). You will not see a mouse cursor and you may close the window by **F4**. In keyboard input mode you may use **TGTERM** or other terminals, which support Esc sequences for non-ASCII keyboard buttons (see "2.12.2 TGTERM terminal").

A little more complicated demo program is given in **HALLOB.C**. The program opens a "Hallo world" window with one "Exit" button, which can be used to close the window.

1.1. Main features

Here is a brief summary of the main 3D GUI features:

- GUI elements are opened by single function calls.
- The package supports mouse input, keyboard input and touch-screen input when working with GUI elements.
- Simple loop for getting GUI events.
- The package supports window dragging (moving).
- The close window function closes all GUI elements, which belong to the current window.

1.2. Restrictions

Here is a brief summary of the 3D GUI restrictions:

- 3D GUI windows are modal. They are organized in a last-in, first-out (LIFO) stack. You are able to work with the last opened window only, called the "active" window. You are able to move the active window only. Return to the parent window by closing the current active window.
- You can't open GUI elements before you open a window.

- Dynamic closing and reopening or modification of opened GUI elements in current active window is not supported.
- Low level drawings in a window are not preserved when moving the window. Use high-level GUI elements only (active or passive).
- Max number of windows to open: 20.
- Total max number of GUI elements in all windows: 300.

1.3. Demo pictures

The 3D GUI distribution archive contains pictures with screen snapshots, showing some GUI applications.

2. Resources

2.1. Coordinate system

The package uses left-handed coordinate system with origin (0,0) at the top left screen corner. All coordinates of GUI elements are relative to this coordinate origin.

2.2. Initialization

You should initialize the 3D GUI package by `tg_init` before calling any other GUI function. Close the package before exit by `tg_exit`. In general, your main function should have the following structure:

```
#include "tg.h"
void main()
{
    TG_RET rc;
    . . . . . .
    rc = tg_init();
    if(rc != RC_OK)
    {
       print("GUI init error = %d\n",(int)rc):
       return;
    }
    . . . . . .    // Call GUI functions, other code ....

    tg_exit();
    return;
}
```

***ATTENTION**. Remember that* `tg_init` *allocates memory. If you forget to call* `tg_exit`*, GUI memory buffers will remain allocated in the camera DRAM and you may encounter memory allocation problems when starting other programs. In this case you need to perform camera power reset.*

2.3. Drawing page

GUI functions draw into current (power-on) overlay page. You can change the current drawing page by `tg_set_drawpage(dr_page)`, where `dr_page` is a start memory address. You can get the start page address by `tg_get_drawpage`.

***ATTENTION**. If you change the default drawing page, you will not see GUI drawings on the camera monitor unless you make current drawing page active by the VCRT function:*

```
setvar(OVLY_START,dr_page);
```

2.4. Drawing origin

All drawings in the GUI package are done relative to current drawing origin, i.e. all pixel coordinates `(x,y)` are added to the coordinates of the origin point. The default drawing origin, set by `tg_init` is

(0,0). Set new drawing origin by tg_set_draworg. Get current drawing origin by tg_get_draworg.

Windows are opened with absolute coordinates regardless to current drawing origin. When you open a window, the drawing origin is set automatically to the upper left corner of the window's client area. All further drawings will be relative to the client area. This is especially important when you open GUI elements, which belong to current window. Remember to save and restore drawing origin if you are going to make custom drawings, followed by opening of GUI element(s).

2.5. Clipping

The GUI functions support pixel coordinate clipping inside default clipping box. The default clipping mode is OFF to achieve higher drawing speed. Use the function tg_set_clipmode to set one of the following clipping modes:

TG_CLIP_OFF = disable coordinate clipping

TG_CLIP_ON = enable normal clipping (draw inside clipping box)

TG_CLIP_INV = enable inverse clipping (draw outside clipping box)

***ATTENTION**. Inverse clipping is not supported by all drawing functions. The following functions support inverse clipping:*

tg_draw_dot

tg_draw_line

tg_draw_box

Use the function tg_set_clipbox to set current clipping box. Setting of clipping box does not change current clipping mode. Get current clipping mode and box by tg_get_clipmode and tg_get_clipbox.

Coordinates of current clipping box are relative to current drawing origin. If you have opened window(s), your drawings by default will be relative to the client's area of the active window.

Tips on clipping:

- The clipping rectangle is relative to the drawing origin.
- Disable clipping to achieve higher drawing speed.
- Some GUI drawing functions use temporary internal clipping. They save current clipping mode and box, set new clipping mode/box, and restore the old mode/box before return. We recommend using the same technique in your custom drawing functions.

2.6. Colors and palettes

The 3D GUI package supports 64 normal colors - 0 to 63. The default color palettes are set by tg_init. The next tables present the available macros and their respective colors.

General colors:

Macro	Color
TG_COLOR_CLEAR	No overlay color
TG_COLOR_BLUE	Blue
TG_COLOR_GREEN	Green
TG_COLOR_CYAN	Cyan

TG_COLOR_RED	Red
TG_COLOR_MAGENTA	Magenta
TG_COLOR_YELLOW	Yellow
TG_COLOR_GRAY	Gray
TG_COLOR_BRIGHT_GRAY	Bright gray
TG_COLOR_BRIGHT_BLUE	Bright blue
TG_COLOR_BRIGHT_GREEN	Bright green
TG_COLOR_BRIGHT_CYAN	Bright cyan
TG_COLOR_BRIGHT_RED	Bright red
TG_COLOR_BRIGHT_MAGENTA	Bright magenta
TG_COLOR_BRIGHT_YELLOW	Bright yellow
TG_COLOR_WHITE	White (high intensity)

Default GUI colors:

Macro	Color
TG_COLOR_BLACK	Black (default foreground text color)
TG_COLOR_DARK_GRAY	Dark-gray (color of shadow 3D corner)
TG_COLOR_DEF_BGND	Middle-gray (windows-like background color)
TG_COLOR_LIGHT_GRAY	Light-gray (color of light 3D corner)
TG_COLOR_TIT_BGND	Background color of inactive window title
TG_COLOR_TIT_SBGND	Background color of active window title

Special transparent colors:

Macro	Color
TG_COLOR_TRANSL_BLUE	Translucent blue (gray picture mixed with blue)
TG_COLOR_TRANSL_GREEN	Translucent green (gray picture mixed with green)
TG_COLOR_TRANSL_RED	Translucent red (gray picture mixed with red)

The number of available 3D GUI colors is defined by the TG_COLOR_CNT macro, defined in **TG.H**. Default colors can be changed by the set-palette function tg_set_palette. Color 0 (TG_COLOR_CLEAR) is reserved and its palette can't be modified.

***ATTENTION**. Be careful when modifying palettes of default GUI colors because you may spoil appearance of GUI elements..*

The color values in the range [0,TG_COLOR_CNT-1] are not stored directly into current drawing overlay page of TI camera. They must be previously encoded by the tg_encode_ticlr function. Example:

```
TG_COLOR clr;
. . . . . .
clr = tg_encode_ticlr(clr);
wpix(clr, ...);     // store clr value into TI overlay page
```

2.7. Fonts

The 3D GUI package supports bitmap fonts with fixed width and height of the character icon. The built-in GUI font has character size 8x10 and contains full table with codes from 0 to 255. The function `tg_set_font` loads new font table from external file in special custom format. The new font table may be full or not, but must always begin from ASCII code 0. The current package version does not support simultaneous usage of multiple font tables.

The function `tg_get_fontsize` returns the size of the current font in pixels. A font editor program exists for creation and modification of font files.

2.8. Box and line definition

The 3D GUI package supports two type definitions for box (rectangle) and line:

`TG_BOX`

`TG_LINE`

These types define 4-element integer buffer, which contains coordinates of two points `(x1,y1)` `(x2,y2)`. Coordinate indexes are accessed by the macros `TG_X1`, `TG_Y1`, `TG_X2` and `TG_Y2`.

The 3D GUI functions work with normalized boxes:

`x1 <= x2` and `y1 <= y2`

i.e. the box must be defined by top/left and bottom/right points in this order. Use the function `tg_norm_box` to normalize a non-normalized box.

Lines are defined similarly. The "normalized" condition is not required.

2.9. Low-level drawing functions

A brief summary of low-level drawing functions is given below. All functions work in current drawing page.

Function	Description
`tg_fill_box`	Fill box with color
`tg_draw_line`	Draw line
`tg_draw_dot`	Draw dot
`tg_read_box`	Read box of pixels
`tg_write_box`	Write box of pixels
`tg_draw_text`	Draw text (basic function)
`tg_draw_box`	Draw box
`tg_draw_textline`	Draw text line with fixed length
`tg_disp_text`	Derivative of tg_draw_text for diag/test purposes
`tg_draw_frame`	Draw frame

2.10. General-purpose utilities

Summary of general-purpose 3D GUI utility functions:

Function	Description
tg_get_screenbox	Get screen box
tg_get_textsize	Get size of displayed text in pixels
tg_copy_box	Copy box
tg_trans_box	Translate box
tg_norm_box	Normalize box
tg_clip_box	Clip box by another box
tg_box_enclose	Check whether one box encloses another box
tg_box_intersect	Check for box intersection
tg_point_inbox	Check whether point is inside box
tg_time	Return system time in ms
tg_wait	Delay program execution in ms
tg_assert	Debug function for TG_ASSERT macro

2.10.1. Utility macros

The header file **TG.H** contains utility macros, which simplify and make more readable your code:

Function	Description
tg_make_box	Fill box coordinates
tg_make_line	Fill line coordinates
tg_box_width	Get box width in pixels
tg_box_height	Get box height in pixels
tg_trans_line	Translate line

2.11. Mouse cursor

Mouse cursor is handled automatically in the "get event" loop (see "3.3 Getting GUI events"). You can hide and show mouse cursor by the following functions:

Function	Description
tg_hide_cursor	hide mouse cursor
tg_show_cursor	show mouse cursor

Normally you don't need to use these functions because the 3D GUI package saves and restores the mouse cursor when necessary. If you write your own drawing functions, you should ensure that mouse cursor is not destroyed. We recommend using the following algorithm:

- Get mouse cursor box by `tg_get_cursbox`.
- Call `tg_box_intersect` to check if cursor box overlaps with your drawing area box.
- If `tg_box_intersect` returns 1 (intersection present), then hide the mouse cursor before drawing and show the mouse cursor after drawing.

2.12. User input

The 3D GUI package works in two modes of user input - mouse and keyboard. The function `tg_set_inputmode` specifies the input mode. Both modes can be set on a platform like PC, which has separate hardware ports for mouse and keyboard. Currently TI cameras have one serial or Ethernet port, which may be used to receive mouse commands or keyboard codes, hence you may use one type of input. The default mode, set by `tg_init`, is keyboard mode.

User input modes (OR of bits):

Function	Description
`TG_INPUT_MODE_MOUSE`	Enable mouse input mode
`TG_INPUT_MODE_KBD`	Enable keyboard input mode

In mouse-input mode the function `tg_set_inputmode` performs initialization of a mouse driver for communication with **attoTerminal** or **TGTERM**. You can get the type of the detected mouse by the `tg_mouse_type` function. Non-zero return value indicates present mouse.

The TI camera has one additional port, called "keypad" port, which may be used to connect a touch-screen device. The keypad is one-directional port – it can receive data only.

2.12.1. attoTerminal

attoTerminal is a Windows terminal with some features, that make it especially useful when working with VC cameras. It supports both TCP and serial connection. You can upload .MSF files to camera and download images from camera.

Using ***attoTerminal*** is fairly simple – just start the program, choose connection method, supply the required connection parameters and click "***Connect***". To enable mouse emulation press **Alt-M**. Second **Alt-M** will stop the mouse emulation.

For more information about ***attoTerminal*** please refer to its manual.

ATTENTION. attoTerminal *does not support Esc sequences for non-ASCII keyboard buttons. This means that you can't use a keyboard for navigation with* ***attoTerminal****. If for some reason you don't intend to work with mouse, consider using TGTERM or other terminal program*

2.12.2. TGTERM terminal

TGTERM is a DOS program, which emulates mouse and keyboard commands for communication with the 3D GUI package. **TGTERM** is extensively tested under Windows 98. You may experience some difficulties when running **TGTERM** with higher Windows versions.

Start **TGTERM** in full DOS screen. Press **`Alt-M`** to enable mouse emulation and start a 3D GUI application with enabled mouse input. Now you can use PC mouse to enter mouse commands.

To work in keyboard-input mode, disable the mouse emulation by second **`Alt-M`** and start 3D GUI application with enabled keyboard input. You can enter normal ASCII codes or non-ASCII key codes. The non-ASCII keys are coded by 3-byte escape sequences:

Key	Byte sequence
`CursorUp`	`Esc [ A`
`CursorDown`	`Esc [ B`
`CursorRight`	`Esc [ C`
`CursorLeft`	`Esc [ D`
`Ins`	`Esc [ I`
`Home`	`Esc [ H`

End	`Esc [ K`
F1	`Esc O q`
F2	`Esc O r`
F3	`Esc O s`
F4	`Esc O t`
F5	`Esc O u`
F6	`Esc O v`
F7	`Esc O w`
F8	`Esc O x`
F9	`Esc O y`
F10	`Esc O p`

Use baud rate 9600 or higher for proper handling of Esc sequences. Esc sequences are converted to valid keyboard event values by the function `tg_get_key` (see "2.13.3.2 Keyboard events").

***INFORMATION**. In keyboard mode you can use other terminals as well - PROCOMM, HyperTerminal, etc., which support Esc sequences for non-ASCII keyboard buttons.*

2.13. High-level GUI functions

High-level GUI functions handle GUI elements. Use high-level GUI functions to:

- Open a GUI element.
- Close window and all GUI elements, which belong to the window.
- Get GUI event (user command, which is processed by some GUI element).
- Get/set GUI data (value associated with the GUI element).

2.13.1. Active GUI elements

Active GUI elements process mouse or keyboard input and return GUI event, which identifies the user command. The current package release supports the following active GUI elements (see function reference for details):

Window:

`tg_WinOpen`	Open window
`tg_WinClose`	Close window and all GUI elements
`tg_WinMainBox`	Get window main box
`tg_WinClientSize`	Get size of client window area

Button:

`tg_ButOpen`	General button open function
`tg_TxtButOpen`	Open text button
`tg_IconButOpen`	Open monochrome or color icon button

Edit:

tg_EditOpen	Open edit GUI element

Spin:

tg_SpinOpen	General spin open function
tg_IntSpinOpen	Open integer spin control
tg_FloatSpinOpen	Open float spin control
tg_ToggleSpinOpen	Open toggle spin control

Radio-button menu:

tg_RadioButOpen	Open radio-button menu

Check-box:

tg_CheckBoxOpen	Open check-box

2.13.2. Passive GUI elements

Passive GUI elements ignore user input. Use these elements to make window drawings, which should be moved together with the window. The current 3D GUI release supports the following passive GUI elements (see function reference for details):

Text

tg_TextOpen	Open text

Frame

tg_FrameOpen	Open frame

Log box

tg_LogBoxOpen	Open log box

2.13.3. Event handling

A set of 3D GUI functions handles input events (see function reference for details):

tg_PumpMouseEvent	Pump mouse event
tg_GetMouseEvent	Get mouse event
tg_ClearMouseEvent	Clear mouse event

tg_PumpKeyEvent	Pump key event
tg_GetKeyEvent	Get key event
tg_ClearKeyEvent	Clear key event

tg_PumpUserInput	Pump mouse and keyboard user input

tg_GetEvent	Main event function, which processes user input and handles all GUI elements in current window

In normal mode of operation (when working with GUI elements of current window), you need to call only tg_GetEvent in a "get event" loop (see "3.3 Getting GUI events"). You can use the other functions in special cases when you need to get user commands in a custom non-GUI function. An example for dragging a custom object, which calls tg_GetMouseEvent, can be found in the demo program **TGBIN.C**.

2.13.3.1. Mouse events

The function tg_GetMouseEvent returns current mouse event into a TG_MOUSE structure with the following members:

event = mouse event (OR of following bits):

TG_MOUSE_LEFT_PRESS	left mouse button press event
TG_MOUSE_LEFT_PRESSED	left mouse button in pressed state
TG_MOUSE_LEFT_REL	left mouse button release event
TG_MOUSE_RIGHT_PRESS	right mouse button pressed
TG_MOUSE_RIGHT_PRESSED	right mouse button in pressed state
TG_MOUSE_RIGHT_REL	right mouse button released
TG_MOUSE_EVENT_NONE	none or processed mouse event

stat = current mouse button press state (OR of the following bits):

TG_MOUSE_BUT_LEFT	0 : left mouse button in released state
	1 : left mouse button in pressed state
TG_MOUSE_BUT_RIGHT	0 : right mouse button in released state
	1 : right mouse button in pressed state

x = absolute x-coord. of mouse cursor

y = absolute y-coord. of mouse cursor

cx = x-coord. of mouse cursor, relative to window client area

cy = y-coord. of mouse cursor, relative to window client area

High-level GUI elements accept the following mouse commands:

Left button press/click	Press/click GUI button
	Set radio-button
	Toggle check-box state on/off
	Select (put on focus) GUI element
Left button continuous pressing	Fast spin increment/decrement
	Drag GUI element (when mouse is moved)

***INFORMATION**. You can use **attoTerminal** or **TGTERM** to enter mouse commands.*

2.13.3.2. Keyboard events

The function `tg_GetKeyEvent` returns a 32-bit key event of type `TG_KEY` in the following format:

`0xN000SSAA` (8 hex digits = 4 bytes)

where:

`N` = flag for available key event

8 : no key event (SSAA = 0000)

0 : key code present in SSAA

`SS` = scan code

`AA` = ASCII code

All ASCII key codes have scan code SS = 00. The header file **TG.H** defines macros for some ASCII and non-ASCII key events:

`TG_KEY_BACKSPACE`	Backspace
`TG_KEY_TAB`	Tab
`TG_KEY_LF`	Line feed
`TG_KEY_ENTER`	Enter (carriage return)
`TG_KEY_ESC`	Esc
`TG_KEY_CURS_UP`	Cursor Up
`TG_KEY_CURS_DOWN`	Cursor Down
`TG_KEY_CURS_RIGHT`	Cursor Right
`TG_KEY_CURS_LEFT`	Cursor Left
`TG_KEY_INS`	Ins
`TG_KEY_DEL`	Del
`TG_KEY_HOME`	Home
`TG_KEY_END`	End
`TG_KEY_F1`	F1
`TG_KEY_F2`	F2
`TG_KEY_F3`	F3
`TG_KEY_F4`	F4
`TG_KEY_F5`	F5
`TG_KEY_F6`	F6
`TG_KEY_F7`	F7
`TG_KEY_F8`	F8
`TG_KEY_F9`	F9
`TG_KEY_F10`	F10

TG_KEY_EVENT_NONE	None or processed key event

High-level GUI elements accept the following key commands:

Tab	Select next window control (put it on focus)
F8	Select previous window control
Enter	Press selected button Set radio-button menu flag Toggle check-box state on/off
Cursor-up	Increment selected spin value Select previous radio-button menu option
Cursor-down	Decrement selected spin value Select next radio-button menu option
F4	Exit window

Key commands used for text editing in the virtual keyboard window:

Esc	Discard changes and restore initial text
Backspace	Delete character before cursor, move cursor left
Ins	Toggle insert mode on/off, change cursor shape
Del	Delete character at cursor position
Home	Move cursor to beginning of edit text
End	Move cursor to end of edit text
Cursor-right	Move cursor one position right
Cursor-left	Move cursor one position left

***INFORMATION**. You can use **TGTERM** to enter keyboard commands.*

2.14. Virtual keyboard

The TI camera has one serial or Ethernet port. You may use the 3D GUI package in mouse or keyboard input mode, but not in both modes. Mouse mode is the preferred mode to work with GUI interface. But using a single mouse you are not able to enter text and numbers in edit and spin GUI elements. The solution of the problem is the so-called "virtual keyboard", which is opened when you click on an edit or spin element with enabled editing.

The virtual keyboard window contains buttons, simulating the buttons of a normal PC keyboard without the numeric pad. An edit box displays the entered text. A spin control selects international keyboard layout.

Enter text by pressing the ASCII buttons (codes 0x20 - 0xFF). Use gray functional buttons to enter editing commands. Note that most keyboard buttons accept continuous pressing for fast button operation. Press the "Enter" button to close the virtual keyboard window. The keyboard text will be returned to the respective GUI element, which has invoked the keyboard.

***ATTENTION**. Don't enable and use the virtual keyboard in keyboard input mode.*

2.14.1. Usage of functional (gray) buttons of the virtual keyboard

Esc	Discard changes and restore initial keyboard text
Backsp	Delete character before cursor, move cursor left
Enter	Close virtual keyboard.
Caps	Toggle capital letters on/off. Button pictures are changed depending on "Caps" state. Second press on "Caps" resets its state.
Shift	Shift next pressed key and then set "Shift" state to "off". Second press on "Shift" resets the shift state.
Ins	Toggle insert mode on/off, change cursor shape
Del	Delete character at cursor position
Home	Move cursor to beginning of edit string
End	Move cursor to end of edit string
Cursor-right	Move cursor one position right
Cursor-left	Move cursor one position left
Up/Down spin buttons	Change international keyboard layout.

***ATTENTION**. You will see initial text in the edit box when you open the virtual keyboard. The initial text will be preserved if the first pressed button is one of the following:*

Backspace, Ins, Del, Home, End, Cursor-right, Cursor-left

*The initial text will be deleted if the first pressed button is an ASCII one. Press **Esc** to restore initial keyboard text at any moment of the editing.*

2.15. Touch-screen

The 3D GUI package supports input from a touch-screen device, based on the AHL-51S "Analog Touch Panel Controller" chip. The touch-screen must perform serial communication at TTL levels (+/-5 V) and should be connected to the keypad port of the TI camera. The touch-screen must be configured in “Make and break” mode.

The touch-screen events are processed concurrently with the mouse events – which comes first is processed first. The touch-screen does not need any software initializations. Note that you can't drag graphical objects by the touch-screen as you are able with the mouse.

The function `ts_calib` performs software calibration of the touch-screen. It clears current drawing page and displays marker at the top left screen corner. Press the touch-screen on the marked point. The function will display a second marker at the bottom right screen corner. Press the touch-screen on second point to complete the calibration. Call this function after `tg_init`, but before opening any window, otherwise you will erase current 3D GUI drawings.

2.16. Copy protection

A registration file protects the 3D GUI package. Send the serial number of your TI camera to the program vendor and you will receive a registration file with name **TGKEY.MSF**. Upload **TGKEY.MSF** to camera flash. Now you will be able to execute programs, linked with the 3D GUI package library **TGLIB.LIB**.

2.17. Memory requirements

The package initialization function allocates memory for system buffers with total size about 25K bytes. Each window open function allocates `dx*dy` bytes, where `dx` and `dy` are the window dimensions. This memory buffer is used to save the screen area below the window. The buffer is released when the window is closed.

Each instance of a GUI element created by respective open function allocates memory buffer with size about 100 bytes.

3. Usage

This chapter describes how to use the 3D GUI package.

3.1. Header file TG.H

The header file **TG.H** contains all definitions and prototypes, needed to call 3D GUI functions. Include the header file in your C source files. Do not use macros and type definitions in **TG.H**, which are not documented in this user's guide. They are for internal use only and may be changed without a notice.

3.2. Opening windows and window controls

The 3D GUI windows are organized in a last-in, first-out (LIFO) stack. The top-of-stack window is active (on focus). The GUI elements in the active window react to user commands, the GUI elements in the inactive windows are disabled. The active window can be moved, the inactive windows can't be moved. When you close the active window, it is popped from the stack and the previous (parent) window becomes active.

GUI elements, opened after a window open operation and before a next window open operation, belong to the window. They are all automatically closed by the window's close function.

Each open function returns a GUI element identifier, which is unique in current window. Use this identifier in a "get event" loop to check if the present GUI event refers to this GUI element and to get/set GUI element data.

GUI functions should be called in the following order:

`tg_WinOpen` : open a new window, which becomes active window (on focus)

`tg_.....Open` : open a GUI element

`tg_.....Open` : open a GUI element

`.................`

ATTENTION. *`tg_WinOpen` sets the drawing origin at the upper left corner of the window's client area - the free window area below the caption. All coordinates of the opened GUI elements are relative to this origin, as well as all drawings, done by the low-level drawing functions. Optional clipping inside the client area can be enabled for test purposes (slower drawing speed). This is valid until a next window is opened, which changes the drawing origin to a new point.*

Example: See "3.7 Examples"

3.3. Getting GUI events

Once you have opened a window and respective GUI elements (controls), you must execute a "get event" loop with the following code:

```
TG_INT16 gui_id;       /* GUI identifier        */
TG_EVENT gui_event;    /* GUI event             */

for(;;)
{
  rc = tg_GetEvent(&gui_id,&gui_event);   // get GUI event
  if(rc != RC_OK) ...;
```

```
  . . . . . .                    // process GUI events (if any)
}
```

The `tg_GetEvent` function checks for user input and updates the appearance of the GUI elements in the active window (if necessary). The function returns two values:

`gui_id` = identifier of GUI element, which has recognized a user command

`gui_event` = type of GUI event (user command)

In general, you should create different "get event" loops in different windows. Remember that current executed "get event" loop refers to current active window. When you close current window, you should return to the previous "get event" loop, where the closed window has been opened.

The execution of the "get event" loop is obligatory because `tg_GetEvent` reads mouse/keyboard commands, updates mouse cursor and handles GUI elements in response to user commands.

Example: See "3.7 Examples"

3.3.1. Types of GUI events

The header file **TG.H** contains macros for GUI events. Use these macros to identify the user commands, which are processed by the GUI elements:

`TG_GUI_EVENT_NONE`	No GUI event
`TG_WIN_EVENT_CLOSE`	Window - close button pressed
`TG_BUT_EVENT_ONFOCUS`	Button put on focus
`TG_BUT_EVENT_PRESS`	Button press event
`TG_BUT_EVENT_PRESSED`	Button in pressed state
`TG_BUT_EVENT_CLICK`	Button release event
`TG_EDIT_EVENT_ONFOCUS`	Edit put on focus
`TG_EDIT_EVENT_CHANGE`	Edit string changed
`TG_SPIN_EVENT_ONFOCUS`	Spin put on focus
`TG_SPIN_EVENT_CHANGE`	Spin value or toggle state changed
`TG_RADIOBUT_EVENT_ONFOCUS`	Radio-button menu put on focus
`TG_RADIOBUT_EVENT_CHANGE`	Radio-button menu state changed
`TG_CHECKBOX_EVENT_ONFOCUS`	Checkbox put on focus
`TG_CHECKBOX_EVENT_CHANGE`	Checkbox state changed

3.4. Getting GUI data

Some GUI elements have associated data. Typical example is the "spin" element - it has current spin value, changed when you press the "up" and "down" buttons. You can get GUI element's data by the `tg_GetGuiData` function. Pass as function argument the GUI identifier, returned by the respective GUI "open" function. The function fills output data buffer and returns the type of the GUI data. You can get the data of the GUI elements in the current active window only.

Types of GUI data:

`TG_GUI_DATA_NONE`	The GUI element has no data
`TG_GUI_DATA_INT`	32-bit integer number

TG_GUI_DATA_FLOAT	Double-precision floating-point number
TG_GUI_DATA_TEXT	Text data (string)

You can get data of the following GUI elements:

GUI element	Data type	Description
Window	TG_GUI_DATA_NONE	No data
Button	TG_GUI_DATA_NONE	No data
Edit	TG_GUI_DATA_TEXT	Get current edit string
Spin	TG_GUI_DATA_INT	Get integer spin value
	TG_GUI_DATA_FLOAT	Get floating-point spin value
	TG_GUI_DATA_INT	Get toggle state
Radio-button	TG_GUI_DATA_INT	Get index of the checked radio-button [0,menu_cnt-1]
Check-box	TG_GUI_DATA_INT	Get 0-not checked, 1-checked
Text	TG_GUI_DATA_NONE	No data
Frame	TG_GUI_DATA_NONE	No data
Log box	TG_GUI_DATA_NONE	No data

There are derivatives of tg_GetGuiData function, which simplify getting of GUI data with known type:

tg_GetTextData	Get text (string) data
tg_GetIntData	Get 32-bit integer data
tg_GetFloatData	Get double floating-point data

3.5. Setting GUI data

You can change GUI element data by the tg_SetGuiData function. Pass as input argument the GUI identifier, returned by the respective GUI "open" function. Note that tg_SetGuiData can set data of passive GUI elements, which can't be read by tg_GetGuiData.

You can set data of the following GUI elements:

GUI element	Data type	Description
Window	TG_GUI_DATA_NONE	No data
Button	TG_GUI_DATA_NONE	No data
Edit	TG_GUI_DATA_TEXT	Set current edit string
Spin	TG_GUI_DATA_INT	Set integer spin value
	TG_GUI_DATA_FLOAT	Set floating-point spin value
	TG_GUI_DATA_INT	Set toggle state
Radio-button	TG_GUI_DATA_INT	Set index of checked radio-button [0, menu_cnt-1]

Check-box	TG_GUI_DATA_INT	Set 0-not checked, 1-checked
Text	TG_GUI_TYPE_TEXT	Change current text
Frame	TG_GUI_TYPE_TEXT	Change frame header text
Log box	TG_GUI_DATA_TEXT	Write text line to log box

3.6. GUI focus

When you open a window with active GUI element(s), the first opened GUI element receives focus. The focus element is marked. In general, the focus shows which GUI element will receive next keyboard command. Mouse commands change focus - the GUI element, which is selected by the mouse cursor, receives the focus. Use the **Tab** key to change focus when you have enabled keyboard input. The selection order is specified by the order of the "open" functions.

Change current focus by `tg_SetFocus`. The function sets focus on a GUI element, specified by a GUI identifier. The function deselects the previous focus element. Passive GUI elements can't receive focus.

Get current focus by `tg_GetFocus`. The function returns the GUI identifier of the current focus element.

3.7. Examples

The following example opens a window and its GUI elements. The code contains exemplary "get event" loop and demonstrates the usage of GUI identifiers, GUI events, and how to get/set GUI data. Some details are omitted intentionally to present the main idea when working with GUI elements.

Refer to "4. Function reference" for more information about parameters of high-level GUI functions.

```
#include "tg.h"
void main()
{
  TG_RET   rc;
  TG_INT16 win_id;      // window GUI identifier
  TG_INT16 but_id;      // button GUI identifier
  TG_INT16 spin_id;     // spin GUI identifier
  TG_INT16 gui_id;      // "get event" GUI identifier
  TG_EVENT gui_event;   // event type

  TG_INT32 spin_val;
  . . . . . .

/*............. Open window and GUI elements */
  rc = tg_WinOpen(..., &win_id);        // open window
  if(rc != RC_OK) ...;
  rc = tg_TxtButOpen(...,&but_id);      // open text button
  if(rc != RC_OK) ...;
  rc = tg_IntSpinOpen(...,&spin_id);    // open integer spin
  if(rc != RC_OK) ...;
  . . . . . .

/*............. Get event loop */
  for(;;)
  {
    rc = tg_GetEvent(&gui_id,&gui_event);              // get GUI event
    if(rc != RC_OK) ...;
    if(gui_event == TG_GUI_EVENT_NONE) continue;       // no event
```

```
/*............. Process window events */
    if(gui_id == win_id)                        // window event present
    {
      if(gui_event == TG_WIN_EVENT_CLOSE)       // close button pressed
      {
        rc = tg_WinClose();     // close current window and its controls
        if(rc != RC_OK) ...;
        break;
      }
    }

/*............. Process button events */
    if(gui_id == but_id)                    // button event detected
    {
      if(gui_event == TG_BUT_EVENT_PRESS)   // button pressed
      {
         . . . . . .                        // process button-press event
      }
    }

/*............. Process spin events */
    if(gui_id == spin_id)                       // spin event detected
    {
      if(gui_event == TG_SPIN_EVENT_CHANGE)     // spin value changed
      {
        rc = tg_GetIntData(spin_id,&spin_val);  // get spin value
        if(rc != RC_OK) ...;
        . . . . . .
        rc = tg_SetGuiData(spin_id,&spin_val);  // set new spin value
        if(rc != RC_OK) ...;
      }
    }

  } /* for(;;) */
}
```

The distribution ZIP file contains source files of several demo programs:

HALLO.C	Simple demo program with "Hallo world..." window
HALLOB.C	Simple demo program with "Hallo world..." window and button
TEV.C	Demo program for GUI events
TGBIN.C	Demo program for picture binarization

3.8. Compiling and linking 3D GUI applications

Compile your code with the TI C/C++ compiler. Link the object modules with the 3D GUI package library **TGLIB.LIB**. We recommend using the following batch file **TCL.BAT** for compilation and linkage of programs with the 3D GUI library (supposing the TI compiler is installed in c:\ti):

```
@echo off
cl6x %1.c -k -o3 -pdr -pdse225 -pl -qq -d_TI_CAMERA
if ERRORLEVEL 1 goto end
lnk6x -q -u _c_int01 %1.obj -m %1.map -o %1.out -l tglib.lib
c:\ti\work\cc.cmd
if ERRORLEVEL 1 goto end

copy %1.out exec.out
c:\ti\util\econv %1
c:\ti\util\scvt
copy adsp.msf %1.msf

:end
```

***ATTENTION**. This batch file requires strict prototyping of all called functions.*

Compile and link demo programs by:

```
tcl tev
tcl tgbin
```

4. Function reference

This chapter describes the 3D GUI library functions in alphabetical order.

4.1. tg_assert - Print assertion message

Prototype:

```
void tg_assert ( char *expr, char *file, unsigned int line )
```

Description:

The function prints message when assertion fails. It is used by the TG_ASSERT macro to print an assertion fail message is in the following format:

TG_ASSERT: ***expr file***(***line***)

Parameters:

expr	Input expression that evaluates to false
file	Input source name
line	Input source file line number

Return code:

None

4.2. tg_box_enclose - Check box enclosing

Prototype:

```
TG_RET tg_box_enclose ( TG_BOX box_en, TG_BOX box )
```

Description:

The function checks whether "box_en" completely encloses "box". The function assumes that the two boxes are normalized:

```
x1 <= x2
y1 <= y2
```

Parameters:

box_en	Input enclosing box
box	Input checked box

Return code:

0	box is not enclosed by box_en
1	box is enclosed by box_en

4.3. tg_box_intersect - Box intersection

Prototype:

```
TG_RET tg_box_intersect ( TG_BOX box1, TG_BOX box2 )
```

Description:

The function checks if there is a common area between the two input boxes and returns respective code.

Parameters:

box1	Input box 1
box2	Input box 2

Return code:

0	No intersection between box1 and box2
1	Intersection present between box1 and box2

4.4. tg_ButOpen - General button open function

Prototype:

```
TG_RET tg_ButOpen ( void *but_buf, short typ, TG_COORD x, TG_COORD y,
TG_COORD dx, TG_COORD dy, short depth, short align, TG_COLOR *clrs,
TG_INT16 *gui_id )
```

Description:

The function opens a button control with position and size, specified by the input parameters.

***ATTENTION**. Button coordinates are relative to the upper left corner of the client area of the last opened window.*

Parameters:

but_buf	Input button picture buffer: string or pixel matrix (NULL: no button picture): type = text : string buffer (char *) type = icon : pointer to TG_IMAGE structure (TG_IMAGE *)
typ	Input button type: TG_BUT_TYPE_TXT : text button TG_BUT_TYPE_ICON : monochrome icon button TG_BUT_TYPE_CLR_ICON : color icon button
x	Input x coordinate of top left button corner (relative to client area of last open window)
y	Input y-coordinate of top left button corner (relative to client area of last open window)
dx	Input button width in pixels
dy	Input button height in pixels
depth	Input button depth (usually 1 or 2)
align	Input horizontal text/icon alignment: TG_ALIGN_LEFT : align left TG_ALIGN_CENTER : align to center TG_ALIGN_RIGHT : align right
clrs	Input buffer with button colors (NULL=use default colors): [0] = bgnd = background color [1] = fgnd = foreground text color [2] = lclr = color of light corner [3] = sclr = color of shadow corner
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
RC_TG_WIN_NOOPEN	No opened window
RC_TG_MALLOC_ERROR	Memory allocation error
Other	Returned by the called functions

4.5. tg_CheckBoxOpen - Open check-box

Prototype:

```
TG_RET tg_CheckBoxOpen ( char *cbox_txt, TG_INT16 cbox_val, TG_COORD x,
TG_COORD y, TG_COLOR *clrs, TG_INT16 *gui_id )
```

Description:

The function opens a check-box control with position, specified by the check-box coordinates (x,y).

***ATTENTION**. Check-box coordinates are relative to the upper left corner of the last opened window client area.*

Parameters:

cbox_txt	Input check-box text
cbox_val	Input initial check-box value: 0 : not set (checked) 1 : set
x	Input x-coordinate of top left check-box corner (relative to the client area of the last opened window)
y	Input y-coordinate of top left check-box corner (relative to the client area of the last opened window)
clrs	Input buffer with check-box colors (NULL: use default colors): [0] = bclr = background check-box color [1] = tclr = foreground check-box color + text color [2] = lclr = color of light check-box corner [3] = sclr = color of shadow check-box corner
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
RC_TG_WIN_NOOPEN	No opened window
RC_TG_MALLOC_ERROR	Memory allocation error
Other	Returned by the called functions

4.6. tg_ClearKeyEvent - Clear key event

Prototype:

```
void tg_ClearKeyEvent ()
```

Description:

The function clears (resets to none) the current key event. This function is called to mark the last key event as processed by some window control. At the same time the key code is not lost - the lower two bytes preserve the code of the the last entered key (see the description of tg_PumpKeyEvent).

Parameters:

None

Return code:

None

4.7. tg_ClearMouseEvent - Clear mouse event

Prototype:

```
void tg_ClearMouseEvent ()
```

Description:

The function clears (resets to none) the current mouse event. The function is called to mark the last mouse event as processed by some window control. At the same time the event code is not lost - the lower bytes of the "event" member keep the bits of the last mouse event (see the description of tg_PumpMouseEvent).

Parameters:

None

Return code:

None

4.8. tg_clip_box - Clip box

Prototype:

```
TG_RET tg_clip_box ( TG_BOX in_box, TG_BOX clip_box, TG_BOX out_box )
```

Description:

The function clips the input box "in_box" by the input clipping box "clip_box" and stores the result box into "out_box".

Parameters:

in_box	Input box
clip_box	Input clipping box
out_box	Output clipped box

Return code:

0	Clipping OK
1	Clipping error: no intersection between in_box and clip_box. The coordinates of out_box are set to -1.

4.9. tg_copy_box - Copy box

Prototype:

```
void tg_copy_box ( TG_BOX in_box, TG_BOX out_box )
```

Description:

The function copies "in_box" into "out_box" (as X/Y coordinates, not as pixel data).

Parameters:

in_box	Input box
out_box	Output box

Return code:

None

4.10. tg_disp_text - Display text (diagnostic function)

Prototype:

```
TG_RET tg_disp_text ( char *str, TG_COORD x, TG_COORD y, TG_COLOR clr, TG_COLOR bclr )
```

Description:

The function displays a text with the specified background and foreground colors. Use this function to display varying text on fixed screen area.

Parameters:

str	Input string with text to display
x	Input x-coordinate of top left text corner
y	Input y-coordinate of top left text corner
clr	Input foreground text color (0 to TG_COLOR_CNT-1)
bclr	Input background color

Return code:

RC_OK	OK
Other	Returned by the called functions

4.11. tg_draw_box - Draw box

Prototype:

```
TG_RET tg_draw_box ( TG_BOX box, int type, TG_COLOR clr, int mode )
```

Description:

The function draws a box (rectangle) in the current drawing page with the specified color and mode. The rectangle is clipped in the current clipping box if clipping is enabled.

***ATTENTION**. This function supports both clipping modes:*
TG_CLIP_ON : normal clipping (draw inside the clip box)
TG_CLIP_INV : inverse clipping (draw outside the clip box).

Parameters:

box	Input box buffer with coordinates of 2 points: (x1,y1) (x2,y2) where: (x1,y1) = top left corner (x2,y2) = bottom right corner Note: The box should be normalized: x1<=x2, y1<=y2
type	Input type of box lines: TG_LINE_SOLID = draw solid box lines

	TG_LINE_DOTTED = draw dotted box lines
clr	Input box color (0 to TG_COLOR_CNT-1)
mode	Input drawing mode: TG_MODE_SET = draw in SET mode TG_MODE_XOR = draw in XOR mode

Return code:

RC_OK	OK
Other	Returned by the called functions

4.12. tg_draw_dot - Draw dot

Prototype:

```
TG_RET tg_draw_dot ( TG_COORD x, TG_COORD y, TG_COLOR clr, int mode )
```

Description:

The function draws a dot in the current drawing page.

***ATTENTION**. The function supports both clipping modes:*
TG_CLIP_ON : normal clipping (draw dot if inside the clip box)
TG_CLIP_INV : inverse clipping (draw dot if outside the clip box)

Parameters:

x	Input dot x-coordinate
y	Input dot y-coordinate
clr	Input dot color (0 to TG_COLOR_CNT-1)
mode	Input drawing mode: TG_MODE_SET = draw in SET mode TG_MODE_XOR = draw in XOR mode

Return code:

RC_OK	OK
Other	Returned by the called functions

4.13. tg_draw_frame - Draw frame

Prototype:

```
TG_RET tg_draw_frame ( char *txt, TG_BOX fbox, TG_COLOR *colors, int depth,
int width, int align, int type )
```

Description:

The function draws 3D frame in the form of a box with an optional header text displayed over the top horizontal frame line. The header text may be centered or aligned left/right as specified by the "align" parameter. The number of lines which produce the 3D view effect (1 or more) is specified by the "depth" parameter. The frame box "fbox" specifies the outer border of the frame. The frame box "fbox" may define one horizontal or vertical 3D line (zero box height or width).

Parameters:

txt	Input frame header text (NULL: no frame header), displayed for frame types TG_FRAME_DOWN_UP and TG_FRAME_UP_DOWN

fbox	Input frame box
colors	Input buffer with frame colors (NULL: use default colors): [0] = bclr = header text background color [1] = tclr = header text foreground color [2] = lclr = color of light frame corner [3] = sclr = color of shadow frame corner
depth	Input frame depth (number of 3D effect lines)
width	Input number of intermediate lines between 3D lines - frame width for types down/up and up/down (can be 0). These lines are not drawn but leaved with current background color.
align	Input header text alignment flag: TG_ALIGN_LEFT : align text left TG_ALIGN_CENTER : align text to center TG_ALIGN_RIGHT : align text right
type	Input frame type: TG_FRAME_DOWN_UP : down & up frame: -\/- TG_FRAME_UP_DOWN : up & down frame: -/\- TG_FRAME_DOWN : down frame: --__ TG_FRAME_UP : up frame: __/--

Return code:

RC_OK	OK
Other	Returned by the called functions

4.14. tg_draw_line - Draw line

Prototype:

```
TG_RET tg_draw_line ( TG_LINE line, int type, TG_COLOR clr, int mode )
```

Description:

The function draws a line in the current drawing page with the specified color and mode. The line is clipped in the current clipping box if clipping is enabled by `tg_set_clipmode`.

***ATTENTION**. This function supports both clipping modes:*
TG_CLIP_ON : normal clipping (draw inside the clip box)
TG_CLIP_INV : inverse clipping (draw outside the clip box)

Parameters:

line	Input line, defined by 2 points (x1,y1) (x2,y2)
type	Input line type: TG_LINE_SOLID = draw solid continuous line TG_LINE_DOTTED = draw dotted line
clr	Input line color (0 to TG_COLOR_CNT-1)
mode	Input drawing mode: TG_MODE_SET = draw in SET mode TG_MODE_XOR = draw in XOR mode

Return code:

RC_OK	OK
RC_TG_MALLOC_ERROR	Memory allocation error

4.15. tg_draw_text - Draw text

Prototype:

```
TG_RET tg_draw_text ( char *str, TG_COORD x, TG_COORD y, TG_COLOR clr,
TG_COLOR bgnd, int mode, int rot )
```

Description:

The function draws text in the current drawing page with the current font. The text is clipped inside the current clipping box if clipping is enabled.

Parameters:

str	Input string with text to display
x	Input x-coordinate of top left text corner
y	Input y-coordinate of top left text corner
clr	Input foreground text color (0 to TG_COLOR_CNT-1)
bgnd	Input background color, used if mode bit TG_DRMODE_BGND_FILL is set to 1
mode	Input drawing mode: bit TG_DRMODE_XOR: 0 : draw in SET mode 1 : draw in XOR mode bit TG_DRMODE_BGND_FILL: 0 : don't fill background color 1 : fill background color bit TG_DRMODE_BIT_MATRIX: 0 : use byte char table (faster) 1 : use bit char table (slower, currently N/A) (mode = TG_DRMODE_DEFAULT : SET mode, no fill, byte matrix)
rot	Input drawing rotation: TG_ROT_0 = 0 degrees (no rotation) TG_ROT_90 = 90 degrees TG_ROT_180 = 180 degrees TG_ROT_270 = 270 degrees (currently N/A)

Return code:

RC_OK	OK
Other	Returned by the called functions

4.16. tg_draw_textline - Draw text line

Prototype:

```
TG_RET tg_draw_textline ( char *txt, TG_COORD x, TG_COORD y, TG_COORD dx,
TG_COLOR clr, TG_COLOR bgnd )
```

Description:

The functions draws a text line with fixed pixel length "dx" in "fill background" mode, i.e. the function draws both foreground text color and fills background color in the text box.

Text-line box:

```
    (x,y)
      +---------------------+------------+ --
```

```
|xxxxxxxxxxxxxxxxxxxx| blank area  |  font_dy
+--------------------+-------------+ --
|<---------- dx pixels ------------>|
|<---- text area ---->|
```

The text is displayed with the current font. If the text is too long to fit in "dx" pixels, it is truncated from right. If the text is shorter than "dx" pixels, the remaining blank area is filled with the background color.

***TIPS & TRICKS**. Use this function to get rid of screen blinking when updating (redrawing) text lines in GUI elements.*

Parameters:

txt	Input text string (NULL: no text to draw, fill only)
x	Input x-coordinate of top left text corner
y	Input y-coordinate of top left text corner
dx	Input width of text line in pixels
clr	Input foreground text color (0 to TG_COLOR_CNT-1)
bgnd	Input background color

Return code:

RC_OK	OK
Other	Returned by the called functions

4.17. tg_EditOpen - Open edit control

Prototype:

```
TG_RET tg_EditOpen ( char *str, int str_size, TG_COORD x, TG_COORD y,
TG_COORD dx, TG_COORD dy, short depth, TG_COLOR *clrs, TG_INT16 *gui_id )
```

Description:

The function opens an edit control with position and size, specified by the input parameters. The function draws the edit control in a 3D box with depth "depth".

***ATTENTION**. The coordinates are relative to the upper left corner of the client area of the last opened window.*

Parameters:

str	Input initial edit string (NULL: empty string)
str_size	Input max length of edited string
x	Input x-coordinate of top left corner of edit box (relative to client area of last open window)
y	Input y-coordinate of top left corner of edit box (relative to client area of last open window)
dx	Input width of edit box in pixels
dy	Input height of edit box in pixels
depth	Input depth of edit box (usually 2)

clrs	Input buffer with edit colors (NULL: use default colors): [0] = bgnd = unselected background color [1] = fgnd = unselected foreground color [2] = lclr = color of light corner [3] = sclr = color of shadow corner [4] = sbgnd = selected background color (on focus) [5] = sfgnd = selected foreground color (on focus)
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
RC_TG_WIN_NOOPEN	No opened window
RC_TG_MALLOC_ERROR	Memory allocation error
Other	Returned by the called functions

4.18. tg_encode_ticlr - Encode TI color

Prototype:

```
TG_COLOR tg_encode_ticlr ( TG_COLOR clr )
```

Description:

The function encodes 3D GUI color value into a pixel value, which should be stored into a TI camera overlay byte.

Parameters:

clr	Input 3D GUI color in the range [0,TG_COLOR_CNT-1]

Return code:

Encoded TI overlay byte.

4.19. tg_exit - Exit 3D GUI package

Prototype:

```
void tg_exit ()
```

Description:

The function exits the 3D GUI package and frees all allocated memory.

Parameters:

None

Return code:

None

4.20. tg_fill_box - Fill box with color

Prototype:

```
TG_RET tg_fill_box ( TG_BOX box, TG_COLOR clr, int mode )
```

Description:

The function sets all pixels in a box to a specified color.

Parameters:

box	Input box
clr	Input pixel color
mode	Input drawing mode: TG_MODE_SET = draw in SET mode TG_MODE_XOR = draw in XOR mode Note: Currently the XOR mode is not implemented on camera.

Return code:

RC_OK	Success
Other	Error codes, returned by the called functions

4.21. tg_FloatSpinOpen - Open floating-point spin control

Prototype:

```
TG_RET tg_FloatSpinOpen ( double spin_val, double spin_step, double spin_min, double spin_max, TG_INT16 edit_mode, TG_INT16 edit_len, TG_COORD x, TG_COORD y, TG_COORD dx, TG_COORD dy, TG_INT16 depth, TG_COLOR *clrs, TG_INT16 *gui_id )
```

Description:

The function opens a spin control with double precision floating-point value.

***ATTENTION**. Spin coordinates are relative to the upper left corner of the client area of the last opened window.*

Parameters:

spin_val	Input initial spin value (double)
spin_step	Input spin increment/decrement step (double)
spin_min	Input minimum spin value (double)
spin_max	Input maximum spin value (double)
edit_mode	Input edit mode: TG_SPIN_EDIT_OFF : disable spin editing TG_SPIN_EDIT_ON : enable spin editing
edit_len	Input max length of edited text in characters (if edit_mode == TG_SPIN_EDIT_ON) and width of edit box in char #
x	Input x-coordinate of top left corner of spin box (relative to client area of last open window)
y	Input y-coordinate of top left corner of spin box (relative to client area of last open window)
dx	Input width of spin box in pixels (edit + buttons)
dy	Input height of spin box in pixels (edit + buttons)
depth	Input depth of spin edit box and spin buttons
clrs	Input buffer with spin colors (NULL: use default colors): [0] = bgnd = unselected edit background color [1] = fgnd = unselected edit foreground color [2] = lclr = color of light corner [3] = sclr = color of shadow corner

	[4] = sbgnd = selected edit background color [5] = sfgnd = selected edit foreground color
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
Other	Returned by the called functions

4.22. tg_FrameOpen - Open frame

Prototype:

```
TG_RET tg_FrameOpen ( char *txt, TG_BOX fbox, TG_COLOR *colors, int depth,
int width, int align, int type, TG_INT16 *gui_id )
```

Description:

The function displays a frame in the current active window. The frame is drawn as a 3D box with optional header text, displayed over the top horizontal border line. The header text may be centered or aligned left/right as specified by the "align" parameter. The number of lines which produce the 3D view effect (1 or more) is specified by the "depth" parameter. The frame box "fbox" specifies the outer border of the frame. The frame box "fbox" may define one horizontal or vertical 3D line (zero box height or width).

ATTENTION.

1. Use this function to draw a frame, which is moved together with the window.

2. Frame coordinates are relative to the upper left corner of the client area of the current active window.

Parameters:

txt	Input frame header text (NULL: no frame header), displayed for frame types TG_FRAME_DOWN_UP and TG_FRAME_UP_DOWN only
fbox	Input frame box
colors	Input buffer with frame colors (NULL: use default colors): [0] = bclr = header text background color [1] = tclr = header text foreground color [2] = lclr = color of light frame corner [3] = sclr = color of shadow frame corner
depth	Input frame depth (number of 3D effect lines)
width	Input number of intermediate lines between 3D lines - frame width for types down/up and up/down (can be 0). Intermediate ines are not drawn but leaved with current background color.
align	Input header text alignment flag: TG_ALIGN_LEFT : align header text left TG_ALIGN_CENTER : align header text to center TG_ALIGN_RIGHT : align header text right
type	Input frame type: TG_FRAME_DOWN_UP : down & up frame: -\/- TG_FRAME_UP_DOWN : up & down frame: -/\- TG_FRAME_DOWN : down frame: --__ TG_FRAME_UP : up frame: __/--
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
RC_TG_MALLOC_ERROR	Memory allocation error
Other	Returned by the called functions

4.23. tg_get_clipbox - Get clipping box

Prototype:

```
void tg_get_clipbox ( TG_BOX box )
```

Description:

The function gets current clipping box, which is used for coordinate clipping by the drawing functions. The default clipping box is the screen (the overlay screen of TI camera).

ATTENTION.

1. The coordinates of the current clipping box are relative to the current drawing origin.
2. Clipping mode (enable/disable) is set by tg_set_clipmode.
3. Disable coordinate clipping to achieve higher drawing speed.

Parameters:

Box	Output clipping box

Return code:

None.

4.24. tg_get_clipmode - Get clipping mode

Prototype:

```
TG_RET tg_get_clipmode ()
```

Description:

The function returns the current clipping mode.

Parameters:

None.

Return code:

TG_CLIP_OFF	clipping disabled
TG_CLIP_ON	normal clipping enabled
TG_CLIP_INV	inverse clipping enabled

4.25. tg_get_cursbox - Get mouse cursor box

Prototype:

```
void tg_get_cursbox ( TG_BOX box )
```

Description:

The function gets current mouse cursor box (absolute coordinates). This box may be used to check if the cursor needs to be saved and restored.

Parameters:

box	Output mouse cursor box

Return code:

None.

4.26. tg_get_draworg - Get drawing origin

Prototype:

```
void tg_get_draworg ( TG_COORD *x, TG_COORD *y )
```

Description:

The function gets the current drawing origin. All drawings in the 3D GUI package are done relative to the drawing origin, i.e. all pixel coordinates (x,y) are added to the coordinates of the current origin point.

Parameters:

x	Output x-coordinate of drawing origin point
y	Output y-coordinate of drawing origin point

Return code:

None.

4.27. tg_get_drawpage - Get drawing page

Prototype:

```
TG_PAGE tg_get_drawpage ()
```

Description:

The function gets current drawing page, used by the drawing functions. On TI camera this is a start byte address of the overlay page.

Parameters:

None.

Return code:

Start byte address of current drawing page.

4.28. tg_get_fontsize - Get font size

Prototype:

```
void tg_get_fontsize ( TG_INT16 *font_dx, TG_INT16 *font_dy )
```

Description:

The function returns in "font_dx" and "font_dy" the width and the height of the current font.

***ATTENTION**. The 3D GUI package supports bitmap fonts with fixed width and height. The built-in font is 8x10 and contains character codes 0-255 (the ASCII table may be full or not).*

Parameters:

font_dx	Output width of font character in pixels
font_dy	Output height of font character in pixels

Return code:

None.

4.29. tg_get_screenbox - Get screen box

Prototype:

```
void tg_get_screenbox ( TG_BOX scr_box )
```

Description:

The function returns in "scr_box" the screen box, detected during the initialization of the 3D GUI package by tg_init().

Parameters:

scr_box	Output screen box

Return code:

None.

4.30. tg_get_textsize - Get text size

Prototype:

```
void tg_get_textsize ( char *str, TG_INT16 *text_dx, TG_INT16 *text_dy )
```

Description:

The function returns in "text_dx" and "text_dy" the dimensions of the screen area, occupied by the text when displayed with current font.

Parameters:

str	Input text string
text_dx	Output width of text area in pixels
text_dy	Output height of text area in pixels

Return code:

None.

4.31. tg_GetEvent - Get GUI event

Prototype:

```
TG_RET tg_GetEvent ( TG_INT16 *gui_id, TG_EVENT *gui_event )
```

Description:

The function processes user input (mouse and/or keyboard commands) for all GUI elements, which belong to the last opened window. These are window controls, which are opened after a window open function and before a next window open function

The type of the detected event is returned in "gui_event". The identifier of the GUI element, which has recognized the user command, is returned in "gui_id".

***ATTENTION**. The last opened window is on focus (selected). After close, the parent window is put on focus.*

Example code for GUI event processing in current window:

```
TG_RET    rc;
```

```
TG_EVENT  gui_event;
TG_INT16  gui_id;
. . . . . .
for(;;)
{
   rc = tg_GetEvent(&gui_id,&gui_event);
   if(rc != RC_OK) ....;               // abort on error
   if(gui_event != TG_GUI_EVENT_NONE)  // GUI event present
   {
      . . . . . .         // process event
   }
}
```

Parameters:

gui_id	Output GUI element identifier
gui_event	Output GUI event type (see TG_GUI_EVENT in tg.h)

Return code:

RC_OK	OK
RC_TG_WIN_INTERNAL_ERR	Internal error
RC_TG_WIN_GUITYPE_ERR	Invalid type of GUI element
RC_TG_INVALID_FOCUS	Invalid id. of focus element
Other	Returned by the called functions

4.32. tg_GetFloatData - Get float GUI data

Prototype:

```
TG_RET tg_GetFloatData ( TG_INT16 gui_id, double *gui_val )
```

Description:

The function reads double floating-point value of a GUI element with identifier "gui_id" in the current window.

Parameters:

gui_id	Input identifier of GUI element: 0 : the window (no data) [1,win_cnt-1] : identifier of window control
gui_val	Output GUI value (floating-point)

Return code:

RC_OK	OK
RC_TG_GUI_DATA_ERR	Invalid type of requested GUI data
Other	Returned by the called functions

4.33. tg_GetFocus - Get focus

Prototype:

```
TG_RET tg_GetFocus ( TG_INT16 *gui_id )
```

Description:

The function gets the window focus. It returns the identifier of the GUI element, which is on focus.

Parameters:

gui_id	Output identifier of GUI element that is on focus: `0 :` the window, no active GUI elements in the window `[1,win_cnt-1] :` id. of GUI element on focus

Return code:

RC_OK	OK
Other	Returned by the called functions

4.34. tg_GetGuiData - Get GUI data

Prototype:

```
TG_RET tg_GetGuiData ( TG_INT16 gui_id, void *data_buf, TG_INT16 data_size,
TG_INT16 *data_typ )
```

Description:

The function reads general data value (with unknown type) of a GUI element with identifier "gui_id" in the current window.

Parameters:

gui_id	Input identifier of GUI element: 0 : the window (no data) [1,win_cnt-1] : identifier of window control
data_buf	Pointer to output buffer, filled with GUI element data
data_size	Input size of data buffer in bytes
data_typ	Output type of GUI element data: TG_GUI_DATA_NONE : GUI element has no data TG_GUI_DATA_INT : 32-bit integer data TG_GUI_DATA_FLOAT : double-precision floating point TG_GUI_DATA_TEXT : text data (string)

Return code:

RC_OK	OK
RC_TG_GUI_DATA_OVF	Overflow of GUI data buffer
RC_TG_GUI_DATA_ERR	Invalid type of GUI data
RC_TG_WIN_GUITYPE_ERR	Invalid type of GUI element (not supported)
Other	Returned by the called functions

4.35. tg_GetIntData - Get integer GUI data

Prototype:

```
TG_RET tg_GetIntData ( TG_INT16 gui_id, TG_INT32 *gui_val )
```

Description:

The function reads integer value of a GUI element with identifier "gui_id" in the current window.

Parameters:

gui_id	Input identifier of GUI element: `0 :` the window (no data) `[1,win_cnt-1] :` identifier of window control

gui_val	Output GUI value (integer)

Return code:

RC_OK	OK
RC_TG_GUI_DATA_ERR	Invalid type of requested GUI data
Other	Returned by the called functions

4.36. tg_GetKeyEvent - Get key event

Prototype:

```
TG_KEY tg_GetKeyEvent ()
```

Description:

The function returns current key event in the following format:

Key code = 0xN000SSAA (8 hex digits = 4 bytes)

where:

N = flag for available key event:

8 : no key event (SSAA = 0000)

0 : key code present in SSAA

SS = scan code

AA = ASCII code

The scan code SS is equal to 00 for all keys with non-zero ASCII codes (AA != 0). The ASCII code AA is equal to 00 for keys, which have scan codes only.

Parameters:

None.

Return code:

Current key event:
0x0000SSAA(>0): Key event code
0x80000000(<0): No key event (TG_KEY_EVENT_NONE)

4.37. tg_GetMouseEvent - Get mouse event

Prototype:

```
void tg_GetMouseEvent ( TG_MOUSE *mouse_event )
```

Description:

The function returns the current mouse event in format, described in tg_PumpMouseEvent.

Parameters:

mouse_event	Output mouse event structure

Return code:

None.

4.38. tg_GetTextData - Get text GUI data

Prototype:

```
TG_RET tg_GetTextData ( TG_INT16 gui_id, char *str, TG_INT16 str_size )
```

Description:

The function reads a text value of a GUI element with identifier "gui_id" in the current window.

Parameters:

gui_id	Input identifier of GUI element: 0 : the window (no data) [1,win_cnt-1] : identifier of window control
str	Output string, filled with GUI element text data
str_size	Input size of "str"

Return code:

RC_OK	OK
RC_TG_GUI_DATA_ERR	Invalid type of requested GUI data
Other	Returned by the called functions

4.39. tg_hide_cursor - Hide mouse cursor

Prototype:

```
void tg_hide_cursor ()
```

Description:

The function hides the mouse cursor by restoring the screen area, specified by the cursor box (the box which encloses the mouse cursor). The screen area is saved when `tg_show_cursor` is called.

Parameters:

None.

Return code:

RC_OK	OK
Other	Returned by the called functions

4.40. tg_IconButOpen Open icon button

Prototype:

```
TG_RET tg_IconButOpen ( TG_IMAGE *icon_img, short icon_typ, TG_COORD x,
TG_COORD y, TG_COORD dx, TG_COORD dy, short depth, short align, TG_COLOR
*clrs, TG_INT16 *gui_id )
```

Description:

The function opens a monochrome or color icon button.

***ATTENTION**. Button coordinates are relative to the upper left corner of the client area of the last opened window.*

Format of icon image buffer "icon_img->img_buf":

Byte matrix with width icon_dx and height icon_dy. Icon pixel rows are stored in "icon_buf" from top to bottom without gaps between sequential rows. Non-zero bytes are drawn with foreground icon color "fgnd" for monochrome icon button. Color icons are drawn with color values, read from "icon_buf" ("fgnd" is ignored).

***ATTENTION**. icon_img->img_pitch should be equal to icon_img->img_dx!*

Parameters:

icon_img	Input icon image with button picture: NULL: no button picture Restriction: img_dx == img_pitch
icon_typ	Input icon button type: TG_BUT_TYPE_ICON : monochrome icon button TG_BUT_TYPE_CLR_ICON : color icon button
x	Input x-coordinate of top left button corner (relative to client area of last opened window)
y	Input y-coordinate of top left button corner (relative to client area of last opened window)
dx	Input button width in pixels
dy	Input button height in pixels
depth	Input button depth (usually 1 or 2)
align	Input horizontal icon alignment: TG_ALIGN_LEFT : align icon left TG_ALIGN_CENTER : align icon to center TG_ALIGN_RIGHT : align icon right
clrs	Input buffer with button colors (NULL=use default colors): [0] = bgnd = background color [1] = fgnd = foreground icon color [2] = lclr = color of light corner [3] = sclr = color of shadow corner
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
Other	Returned by the called functions

4.41. tg_init - Initialize 3D GUI package

Prototype:

```
TG_RET tg_init ()
```

Description:

The function performs the following 3D GUI-package initialization operations:
1. Sets default drawing page.
2. Gets screen size.
3. Sets default drawing origin.
4. Sets default clipping mode and clipping box.
5. Initializes colors.
The TI camera color overlay is initialized to all 8 bit-planes. The camera displays simultaneously maximum 67 different overlay colors:

- no color ("clear" color), set by overlay byte == 0
- 63 normal colors
- 3 translucent colors

6. Sets default color palette for each 3D GUI color.
7. Loads default font table.
8. Sets other 3D GUI parameters.

Parameters:

None.

Return code:

RC_OK	OK
RC_TG_INIT_ERROR	Unable to initialize 3D GUI package
RC_TG_MALLOC_ERROR	Memory allocation error
Other	Returned by the called functions

4.42. tg_IntSpinOpen - Open integer spin control

Prototype:

```
TG_RET tg_IntSpinOpen ( TG_INT32 spin_val, TG_INT32 spin_step, TG_INT32 spin_min, TG_INT32 spin_max, TG_INT16 edit_mode, TG_INT16 edit_len, TG_COORD x, TG_COORD y, TG_COORD dx, TG_COORD dy, TG_INT16 depth, TG_COLOR *clrs, TG_INT16 *gui_id )
```

Description:

The function opens a spin control with integer value.

***ATTENTION**. Spin coordinates are relative to upper left corner of client area of last opened window.*

Parameters:

spin_val	Input initial spin value (32-bit integer)
spin_step	Input spin increment/decrement step (32-bit integer)
spin_min	Input minimum spin value (32-bit integer)
spin_max	Input maximum spin value (32-bit integer)
edit_mode	Input edit mode: TG_SPIN_EDIT_OFF : disable spin editing TG_SPIN_EDIT_ON : enable spin editing
edit_len	Input max length of edited text in characters (if edit_mode == TG_SPIN_EDIT_ON) and width of edit box in char #
x	Input x-coordinate of top left corner of spin box (relative to client area of last open window)
y	Input y-coordinate of top left corner of spin box (relative to client area of last open window)
dx	Input width of spin box in pixels (edit + buttons)
dy	Input height of spin box in pixels (edit + buttons)
depth	Input depth of spin edit box and spin buttons
clrs	Input buffer with spin colors (NULL: use default colors): [0] = bgnd = unselected edit background color

	[1] = fgnd = unselected edit foreground color [2] = lclr = color of light corner [3] = sclr = color of shadow corner [4] = sbgnd = selected edit background color [5] = sfgnd = selected edit foreground color
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
Other	Returned by the called functions

4.43. tg_LogBoxOpen - Open log box

Prototype:

```
TG_RET tg_LogBoxOpen ( TG_COORD x, TG_COORD y, TG_COORD dx, TG_COORD dy,
short depth, TG_COLOR *clrs, TG_INT16 *gui_id )
```

Description:

The function opens a log box with position and size, specified by the input parameters. The log box resembles a monitor screen, which displays text lines. Each new text line is displayed on the bottom screen line. The old text lines are scrolled up to free the bottom line.

The function draws 3D log box with depth "depth", which is initially empty (no log text). Use tg_SetGuiData() function to write a text line into the log box.

ATTENTION.

1. The log box moves together with the window.
2. Log box coordinates are relative to the upper left corner of the client area of the current active window.

Parameters:

x	Input x-coordinate of top left corner of log box
y	Input y-coordinate of top left corner of log box
dx	Input width of log box in pixels
dy	Input height of log box in pixels
depth	Input depth of log box (usually 1 or 2)
clrs	Input buffer with log box colors (NULL: use default colors): [0] = bgnd = background color [1] = fgnd = foreground text color [2] = lclr = color of light corner [3] = sclr = color of shadow corner
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
RC_TG_MALLOC_ERROR	Memory allocation error
Other	Returned by the called functions

4.44. tg_norm_box - Normalize box

Prototype:

```
void tg_norm_box ( TG_BOX in_box, TG_BOX out_box )
```

Description:

The function normalizes "in_box" and stores the result box into "out_box". Normalized boxes have the following coordinates:

```
x1 <= x2
y1 <= y2
```

The normalization can be done in place (in_box == out_box).

Parameters:

in_box	Input box
out_box	Output box

Return code:

None.

4.45. tg_point_inbox - Check for point inside box

Prototype:

```
TG_RET tg_point_inbox ( TG_BOX box, TG_COORD x, TG_COORD y )
```

Description:

The function checks whether the point (x,y) is inside "box".

Parameters:

box	Input box
x	Input x-coordinate of tested point
y	Input y-coordinate of tested point

Return code:

0	Point outside box
1	Point inside box

4.46. tg_PumpKeyEvent - Pump key event

Prototype:

```
void tg_PumpKeyEvent ()
```

Description:

The function checks for a key event (a pressed key) and sets a global 32-bit key code in the following format:

Key code = `0xN000SSAA` (8 hex digits = 4 bytes)

where:

`N` = flag for available key event:

`8` : no key event (`SSAA = 0000`)

`0` : key code present in SSAA

SS = scan code

AA = ASCII code

The scan code SS is equal to 00 for all keys with non-zero ASCII codes (`AA != 0`). The ASCII code AA is equal to `00` for keys, which have scan codes only.

Parameters:

None.

Return code:

None.

4.47. tg_PumpMouseEvent - Pump mouse event

Prototype:

```
TG_RET tg_PumpMouseEvent ()
```

Description:

The function checks for a mouse event and sets global mouse event structure with the following members:

`event`	mouse event (OR of event bits):	
	bit TG_MOUSE_LEFT_PRESS	: left button press event
	bit TG_MOUSE_LEFT_PRESSED	: left button in pressed state
	bit TG_MOUSE_LEFT_REL	: left button release event
	bit TG_MOUSE_RIGHT_PRESS	: right button press event
	bit TG_MOUSE_RIGHT_PRESSED	: right button in pressed state
	bit TG_MOUSE_RIGHT_REL	: right button release event
	bit TG_MOUSE_EVENT_NONE	: no mouse event
`stat`	current button press state	
`x`	x-coordinate of mouse cursor hot spot (absolute)	
`y`	y-coordinate of mouse cursor hot spot (absolute)	

Mouse coordinates, relative to the client area of the current window:

`cx`	relative x-coordinate of mouse cursor
`cy`	relative y-coordinate of mouse cursor

***ATTENTION**. These coordinates are initially equal to (x,y). They are converted to relative by the tg_GetEvent function. The function re-draws mouse cursor when necessary.*

Parameters:

None.

Return code:

RC_OK	Success
TG_MOUSE_OFFLINE	No connected mouse
RC_TG_MOUSE_ERROR	Mouse I/F error
Other	Error codes, returned by the called functions

4.48. tg_PumpUserInput - Pump user input

Prototype:

```
TG_RET tg_PumpUserInput ()
```

Description:

The function checks for mouse and key user-input and fills corresponding global event buffers.

Parameters:

None.

Return code:

RC_OK	Success
Other	Error codes, returned by the called functions

4.49. tg_RadioButOpen - Open radio-button menu

Prototype:

```
TG_RET tg_RadioButOpen ( char **menu_txt, TG_UINT16 menu_cnt, TG_INT16
menu_val, TG_INT16 type, TG_INT16 dist, TG_COORD x, TG_COORD y, TG_COLOR
*clrs, TG_INT16 *gui_id )
```

Description:

The function opens a radio-button menu with position, specified by the menu coordinates (x,y).

ATTENTION. *Menu coordinates are relative to the upper left corner of the last opened window client area.*

Parameters:

menu_txt	Input buffer with radio-button menu texts
menu_cnt	Input number of menu items (should be >= 2) (use check-box for 1-item menu)
menu_val	Input index of initial active radio-button in the range [0,menu_cnt-1]
type	Input type of radio-button menu: TG_RADIOBUT_TYPE_VERT : vertical TG_RADIOBUT_TYPE_HORIZ : horizontal
dist	Input distance between menu items in pixels
x	Input x-coord. of top left corner of radio-button menu (relative to client area of last open window)
y	Input y-coord. of top left corner of radio-button menu (relative to client area of last open window)
clrs	Input buffer with radio-button menu colors (NULL: use default colors): [0] = bclr = bgnd radio-button color [1] = tclr = fgnd radio-button + text color [2] = lclr = color of light corner [3] = sclr = color of shadow corner
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
RC_TG_WIN_NOOPEN	No opened window
RC_TG_INVALID_ARG	Invalid argument: menu_cnt <
RC_TG_MALLOC_ERROR	Memory allocation error
Other	Returned by the called functions

4.50. tg_read_box - Read box

Prototype:

```
TG_RET tg_read_box ( TG_BOX box, char *buf )
```

Description:

The function reads a box of pixels from current drawing page into the output buffer "buf". Pixels are read from top to bottom box row, starting from the top left box corner. No gaps should occur between sequential pixel rows.

ATTENTION. *If clipping is enabled and "box" is clipped, then parts of "buf" are not filled with pixel values.*

Parameters:

box	Input box
buf	Output pixel buffer (size = `box_dx * box_dy` bytes)

Return code:

RC_OK	Success
Other	Error codes, returned by the called functions

4.51. tg_set_clipbox - Set clipping box

Prototype:

```
void tg_set_clipbox ( TG_BOX box )
```

Description:

The function sets current clipping box, which is used for coordinate clipping by the drawing functions. The default clipping box is the screen (the overlay screen of TI camera).

ATTENTION.

1. Coordinates of the current clipping box are relative to the current drawing origin.
2. Clipping mode (enable/disable) is set by tg_set_clipmode.
3. Disable coordinate clipping to achieve higher drawing speed.

Parameters:

box	Input clipping box

Return code:

None.

4.52. tg_set_clipmode - Set clipping mode

Prototype:

```
void tg_set_clipmode ( int clip )
```

Description:

The function enables or disables coordinate clipping inside current clipping box, which is done by the low-level drawing functions. The default clipping mode is OFF to achieve higher drawing speed.

ATTENTION.

1. Coordinates of the current clipping box are relative to the drawing origin.
2. Inverse clipping is not supported by all low-level drawing functions.

Parameters:

clip	Input clipping mode: TG_CLIP_OFF = disable coordinate clipping TG_CLIP_ON = normal clipping (draw inside clip box) TG_CLIP_INV = inverse clipping (draw outside clip box)

Return code:

None.

4.53. tg_set_draworg - Set drawing origin

Prototype:

```
void tg_set_draworg ( TG_COORD x, TG_COORD y )
```

Description:

The function sets current drawing origin. All drawings in the 3D GUI package are done relative to the drawing origin, i.e. all pixel coordinates (x,y) are added to coordinates of the current origin point.

Parameters:

x	Input x-coordinate of drawing origin point
y	Input y-coordinate of drawing origin point

Return code:

None.

4.54. tg_set_drawpage - Set drawing page

Prototype:

```
void tg_set_drawpage ( TG_PAGE dr_page )
```

Description:

The function sets current drawing page, used by the drawing functions. On TI camera this is the start byte address of the overlay page.

Parameters:

dr_page	Input drawing page (start address of TI overlay page)

Return code:

None.

4.55. tg_set_font - Set font

Prototype:

```
TG_RET tg_set_font ( char *font_file )
```

Description:

The function installs new font to be used by the 3D GUI package functions. The font file must be in a bitmap format, supported by the 3D GUI package. In case of font error, the function leaves current font active.

ATTENTION. *The present 3D GUI package release does not supports different fonts for different GUI elements in one window. We recommend to use one font, set immediately after tg_init().*

Parameters:

font_file	Input font file name

Return code:

RC_OK	Success
RC_TG_FONT_FILE_ERR	Font file error
RC_TG_INVALID_FONT	Font format error
RC_TG_MALLOC_ERROR	Font table allocation error
Other	Returned by the called functions

4.56. tg_set_inputmode - Set input mode

Prototype:

```
TG_RET tg_set_inputmode ( TG_INT16 mode )
```

Description:

The function specifies mode for user input: mouse and/or keyboard. Both modes can be set concurrently on a hardware platform like PC, which has separate input ports for mouse and keyboard. One mode must be set on TI camera, which receives mouse commands or keyboard strokes through the serial/Telnet port.

ATTENTION. *The function performs initialization of the mouse driver. Similar initialization for keyboard or other device should be done if necessary.*

Parameters:

mode	Input user-input mode (OR of bits): bit TG_INPUT_MODE_MOUSE : mouse input mode bit TG_INPUT_MODE_KBD : keyboard input mode

Return code:

RC_OK	Success
Other	Returned by the called functions

4.57. tg_set_palette - Set color palette

Prototype:

```
void tg_set_palette ( TG_COLOR clr, TG_COLOR r, TG_COLOR g, TG_COLOR b )
```

Description:

The function sets RGB palette for input color. The color must be in the range of available colors [0,TG_COLOR_CNT-1], otherwise the function does nothing.

ATTENTION. *Setting palette for color 0 (clear) is ignored on TI camera.*

Parameters:

clr	Input color
r	Input red intensity [0,255]
g	Input green intensity [0,255]
b	Input blue intensity [0,255]

Return code:

None.

4.58. tg_SetFocus - Set focus

Prototype:

```
TG_RET tg_SetFocus ( TG_INT16 gui_id )
```

Description:

The function sets focus on the GUI element "gui_id" in the current window. If the focus is changed, it deselects the previous focus element and selects the "gui_id" element. Selecting and deselecting means redrawing the element.

ATTENTION. *Focus can be changed for non-passive GUI elements, which process user-input events.*

Parameters:

gui_id	Input identifier of GUI element that is put on focus: 0 : the window itself: do nothing [1,win_cnt-1] : put on focus "gui_id" element

Return code:

RC_OK	OK
RC_TG_WIN_GUITYPE_ERR	Invalid type of GUI element
RC_TG_INVALID_FOCUS	Invalid id. of focus element
Other	Returned by the called functions

4.59. tg_SetGuiData - Set GUI data

Prototype:

```
TG_RET tg_SetGuiData ( TG_INT16 gui_id, void *data_buf )
```

Description:

The function sets new data value of the GUI element with identifier "gui_id" in the current window. The data buffer "data_buf" **MUST** contain value, which coincides with the value type of the GUI element.

Parameters:

gui_id	Input identifier of GUI element: 0 : the window (no data) [1,win_cnt-1] : identifier of window control
data_buf	Input buffer with new data of GUI element

Return code:

RC_OK	OK
RC_TG_WIN_GUITYPE_ERR	Invalid type of GUI element (not supported)
Other	Returned by the called functions

4.60. tg_show_cursor - Show mouse cursor

Prototype:

```
TG_RET tg_show_cursor ()
```

Description:

The function shows the mouse cursor. Before drawing the cursor it generates a mouse cursor box (the box which encloses the cursor) and saves the screen area, specified by the cursor box.

Parameters:

None.

Return code:

RC_OK	OK
Other	Returned by the called functions

4.61. tg_SpinOpen - General spin open function

Prototype:

```
TG_RET tg_SpinOpen ( TG_SPIN_PAR *spin_par, TG_COORD x, TG_COORD y,
TG_COORD dx, TG_COORD dy, TG_INT16 depth, TG_COLOR *clrs, TG_INT16 *gui_id
)
```

Description:

The function opens a spin control with position and size, specified by the input parameters. Some spin-open parameters are passed by the input parameter structure "spin_par" with the following members:

`spin_val`	Input ptr to initial spin value
`spin_step`	Input ptr to spin increment/decrement step
`spin_min`	Input ptr to minimum spin value
`spin_max`	Input ptr to maximum spin value
`spin_type`	Input type of spin value: TG_GUI_DATA_INT : 32-bit integer TG_GUI_DATA_FLOAT : double-precision float TG_GUI_DATA_TEXT : toggle (use toggle arguments)
`toggle_buf`	Input buffer with toggle strings, used when: `spin_type == TG_GUI_DATA_TEXT`

`toggle_cnt`	Input number of toggle states (size of toggle_buf)
`toggle_state`	Input initial toggle state in the range `[0,toggle_cnt-1]` - initial selected toggle string
`edit_str_size`	Input max length of spin edit string in char #, used when: `spin_edit_mode == TG_SPIN_EDIT_ON`
`spin_edit_mode`	Input edit mode: TG_SPIN_EDIT_OFF : disable spin editing TG_SPIN_EDIT_ON : enable spin editing

ATTENTION. *Spin coordinates are relative to the upper left corner of the client area of the last opened window.*

Parameters:

spin_par	Input spin parameter structure (see tg.h).
x	Input x-coordinate of top left corner of spin box (relative to client area of last open window)
y	Input y-coordinate of top left corner of spin box (relative to client area of last open window)
dx	Input width of spin box in pixels (edit + buttons)
dy	Input height of spin box in pixels (edit + buttons)
depth	Input depth of spin edit box and spin buttons
clrs	Input buffer with spin colors (NULL: use default colors): [0] = bgnd = unselected edit background color [1] = fgnd = unselected edit foreground color [2] = lclr = color of light corner [3] = sclr = color of shadow corner [4] = sbgnd = selected edit background color (on focus) [5] = sfgnd = selected edit foreground color (on focus)
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
RC_TG_WIN_NOOPEN	No opened window
RC_TG_MALLOC_ERROR	Memory allocation error
RC_TG_INVALID_TYPE	Invalid type of spin value
Other	Returned by the called functions

4.62. tg_TextOpen - Open text

Prototype:

```
TG_RET tg_TextOpen ( char *text, TG_COORD x, TG_COORD y, TG_COORD dx,
TG_COLOR clr, TG_COLOR bgnd, TG_INT16 *gui_id )
```

Description:

The function displays text in the current active window.

***TIPS & TRICKS**. Use this function to draw a text inside a window, which should be moved together with the window.*

***ATTENTION.** Text coordinates are relative to upper left corner of the client area of the current active window.*

Parameters:

text	Input text to display
x	Input x-coordinate of top left text corner
y	Input y-coordinate of top left text corner
dx	Input text length in pixels: <=0 : automatic length, display all text characters
clr	Input foreground text color
bgnd	Input background text color: TG_COLOR_CLEAR : don't draw bgnd color
gui_id	Output GUI element identifier (NULL: don't return)
text	Input text to display

Return code:

RC_OK	OK
RC_TG_MALLOC_ERROR	Memory allocation error
Other	Returned by the called functions

4.63. tg_time - Get time

Prototype:

```
unsigned long tg_time ()
```

Description:

The function returns the system time in milliseconds. The time grows up.

Parameters:

None.

Return code:

System time (ms).

4.64. tg_ToggleSpinOpen - Open toggle spin control

Prototype:

```
TG_RET tg_ToggleSpinOpen ( char **toggle_buf, TG_INT16 toggle_cnt, TG_INT16
toggle_state, TG_COORD x, TG_COORD y, TG_COORD dx, TG_COORD dy, TG_INT16
depth, TG_COLOR *clrs, TG_INT16 *gui_id )
```

Description:

The function opens a “toggle”-type spin control. The toggle strings are selected consecutively in forward order by the “down” button or reverse order by the “up” spin button.. Editing is disabled. The spin value is defined as the current toggle state – the index of the selected toggle string in the range `[0 to toggle_cnt-1]`.

ATTENTION. *Spin coordinates are relative to upper left corner of the client area of the last opened window.*

Parameters:

toggle_buf	Input buffer with toggle strings
toggle_cnt	Input number of toggle states (size of toggle_buf)
toggle_state	Input initial toggle state in range [0,toggle_cnt-1]
x	Input x-coordinate of top left corner of spin box (relative to client area of last open window)
y	Input y-coordinate of top left corner of spin box (relative to client area of last open window)
dx	Input width of spin box in pixels (edit + buttons)
dy	Input height of spin box in pixels (edit + buttons)
depth	Input depth of spin edit box and spin buttons
clrs	Input buffer with spin colors (NULL: use default colors): [0] = bgnd = unselected edit background color [1] = fgnd = unselected edit foreground color [2] = lclr = color of light corner [3] = sclr = color of shadow corner [4] = sbgnd = selected edit background color [5] = sfgnd = selected edit foreground color
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
Other	Returned by the called functions

4.65. tg_trans_box - Translate box

Prototype:

```
void tg_trans_box ( TG_BOX box, TG_COORD dx, TG_COORD dy )
```

Description:

The function adds horizontal and vertical displacements dx and dy to the coordinates of the input/output box "box".

Parameters:

box	Input/output box

dx	Input x-coordinate displacement
dy	Input y-coordinate displacement

Return code:

None.

4.66. tg_TxtButOpen - Open text button

Prototype:

```
TG_RET tg_TxtButOpen ( char *txt, TG_COORD x, TG_COORD y, TG_COORD dx,
TG_COORD dy, short depth, short align, TG_COLOR *clrs, TG_INT16 *gui_id )
```

Description:

The function opens a text button with position and size, specified by the input parameters.

ATTENTION. *Button coordinates are relative to upper left corner of client area of last opened window.*

Parameters:

txt	Input string with button text (NULL: no button text)
x	Input x-coordinate of top left button corner (relative to client area of last opened window)
y	Input y-coordinate of top left button corner (relative to client area of last opened window)
dx	Input button width in pixels
dy	Input button height in pixels
depth	Input button depth (usually 1 or 2)
align	Input horizontal text alignment: TG_ALIGN_LEFT : align text left TG_ALIGN_CENTER : align text to center TG_ALIGN_RIGHT : align text right
clrs	Input buffer with button colors (NULL=use default colors): [0] = bgnd = background color [1] = fgnd = foreground text color [2] = lclr = color of light corner [3] = sclr = color of shadow corner

Return code:

RC_OK	OK
Other	Returned by the called functions

4.67. tg_wait - Wait (ms)

Prototype:

```
void tg_wait ( int wait_time )
```

Description:

The function waits "wait_time" milliseconds (maximum 1 second).

Parameters:

wait_time	Input wait time in ms (< 1000)

Return code:

None.

4.68. tg_WinClientSize - Get size of client window area

Prototype:

```
TG_RET tg_WinClientSize ( TG_COORD *client_dx, TG_COORD *client_dy )
```

Description:

The function returns the width and the height of the client area of the current active window (on focus).

Parameters:

client_dx	Output width of window client area
client_dy	Output height of window client area

Return code:

RC_OK	OK
RC_TG_WIN_NOOPEN	No opened window
Other	Returned by the called functions

4.69. tg_WinClose - Close window

Prototype:

```
TG_RET tg_WinClose ()
```

Description:

The function closes the last opened window and its GUI elements and returns to the parent window. The function frees all memory, allocated by the last window and its elements.

Parameters:

None.

Return code:

RC_OK	OK
RC_TG_WIN_INTERNAL_ERR	Internal window error
RC_TG_WIN_GUITYPE_ERR	Unknown type of GUI element
Other	Returned by the called functions

4.70. tg_WinMainBox - Get window main box

Prototype:

```
TG_RET tg_WinMainBox ( TG_BOX box )
```

Description:

The function returns in "box" the main box of the current active window.

Parameters:

box	Output main window box (absolute coordinates)

Return code:

RC_OK	OK
RC_TG_WIN_NOOPEN	No opened window
Other	Returned by the called functions

4.71. tg_WinOpen - Open window

Prototype:

```
TG_RET tg_WinOpen ( char *title, TG_COORD x, TG_COORD y, TG_COORD dx,
TG_COORD dy, TG_INT16 attr, TG_INT16 depth, TG_INT16 *gui_id )
```

Description:

The function opens a window. All GUI elements, opened after the last window-open operation, belong to this window. Their coordinates are relative to upper left corner of the window's client area. Windows are opened with absolute screen coordinates.

ATTENTION. *Window titles are displayed with default background and foreground colors (active and inactive).*

Parameters:

title	Input window title string (NULL: no window title)
x	Input x-coordinate of top left window corner (absolute)
y	Input y-coordinate of top left window corner (absolute)
dx	Input window width in pixels
dy	Input window height in pixels
attr	Input window attributes (OR of bits): 0 : no close button TG_WIN_CLOSE_BUT : enable close button
depth	Input window depth (also depth of close button)
gui_id	Output GUI element identifier (NULL: don't return)

Return code:

RC_OK	OK
RC_TG_MALLOC_ERROR	Memory allocation error
Other	Returned by the called functions

4.72. tg_write_box - Write box

Prototype:

```
TG_RET tg_write_box ( TG_BOX box, char *buf )
```

Description:

The function writes a box of pixels into current drawing page. Pixels are read from the input buffer "buf" from top to bottom box row, starting from the upper left box corner, without gaps between sequential pixel rows.

Parameters:

box	Input box
buf	Input pixel buffer (size = `box_dx * box_dy` bytes)

Return code:

RC_OK	Success
Other	Error codes, returned by the called functions

4.73. ts_calib - Calibrate touch-screen

Prototype:

```
TG_RET ts_calib ( TG_PAGE dr_page, int clr )
```

Description:

The function calibrates the touch-screen in the following steps:

1. Clears current drawing page.
2. Displays a marker at the top left screen corner and prompts you to press the marked point.
3. Displays a marker at the bottom right screen corner and prompts you to press the marked point.

Parameters:

dr_page	Input drawing page (start address of TI overlay page)
clr	Input drawing color for marker and message texts

Return code:

RC_OK	Success
Other	Returned by called functions

5. Error codes

The 3D GUI functions return the following error codes:

Code	Macro	Description
4101	RC_TG_INIT_ERROR	3D GUI package initialization error
4102	RC_TG_MALLOC_ERROR	Memory allocation error
4103	RC_TG_MOUSE_ERROR	Mouse state error (I/O error)
4104	RC_TG_GUI_STACK_OVF	GUI stack overflow
4105	RC_TG_GUI_STACK_ACC_ERR	GUI stack access error
4106	RC_TG_WIN_SIZE_ERROR	Window size error (outside screen)
4107	RC_TG_WIN_STACK_OVF	Window stack overflow
4108	RC_TG_WIN_STACK_ACC_ERR	Window stack access error
4109	RC_TG_WIN_INTERNAL_ERR	Internal window error
4110	RC_TG_WIN_GUITYPE_ERR	Invalid type of GUI element
4111	RC_TG_WIN_NOOPEN	No opened window
4112	RC_TG_GUI_DATA_OVF	Overflow of GUI data buffer
4113	RC_TG_GUI_DATA_ERR	Invalid type of GUI data
4114	RC_TG_INVALID_ARG	Invalid argument passed to function
4115	RC_TG_INVALID_TYPE	Invalid type
4116	RC_TG_FONT_FILE_ERR	Font file error
4117	RC_TG_INVALID_FONT	Invalid font
4118	RC_TG_MEMBUF_OVF	Memory buffer overflow
4119	RC_TG_INVALID_BOX	Invalid box
4120	RC_TG_INVALID_FOCUS	Invalid id. of focus element
4121	RC_TG_KBD_TYPE_ERROR	Virtual keyboard type not supported
4122	RC_TG_KBD_MISMATCH_ERR	Virtual keyboard mismatch error
4123	RC_TG_KBD_OPEN_ERR	Virtual keyboard open error
4124	RC_TG_KEYPAD_FIFO_OVF	Keypad FIFO buffer overflow
4125	RC_TG_KEYPAD_INT_ERR	Keypad internal error
4126	RC_TG_TOUCH_FMT_ERR	Touch-screen data format error
4127	RC_TG_INVALID_KEY	Invalid/missing registration key