Freescale Embedded GUI (D4D)

Document Number: DRM116
Rev. 2
10/2010

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 26668334
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor

DRM116
Rev. 2
10/2010

Chapter 1
Introduction

Chapter 2
Description

Chapter 3
Driver API

Chapter 1
Introduction

Many embedded applications require a human interface for control and display purposes, but until recently these have been restricted to simple led or segment LCD displays for most MCU applications due to the limited RAM and ROM available on the chip. When the required complexity has led to the need for a graphics LCD panel, then the hardware would normally be changed to add an additional MPU with a dedicated LCD drive or with the addition of an external graphics LCD controller. The additional cost associated with this has restricted the introduction of graphics LCDs to most MCU applications. However, with the recent introduction of low-priced graphics LCD panels with simple serial or parallel interfaces and integrated display RAM, graphic LCD applications can now easily be implemented with MCUs if the graphics driver software is well designed and takes into consideration the limited MCU resources. The D4D has been specifically written with the constraints of an MCU (low FLASH and RAM) and the assumption that the graphics display RAM is write-only, as is the case of many “Smart LCD” panels. As a result, the D4D can produce a stunning layered graphics display using only limited RAM and FLASH from the MCU and a small library footprint.

All the code in this application note has been tested and debugged on the MC9S08QE128, MCF51QE128, MCF51JM128, MCF51CN128, MCF52259, MCF52277, and MPC5125 with CodeWarrior for Microcontrollers v6.3. for HCS08 MCU's, CodeWarrior for Coldfire v7.2 and CodeWarrior for MobileGT 9.2.

1.1 Display Module

The display driver and its API is tested on the Display3000 color display modules installed on the DemoQE development board and TOWER LCD module with ColdFire CV1 and CV2. The MCF52277 was tested on the MCF52277EVB on the LCD board. The MPC5125 was tested on the PC digital LCD display up to 1024 x 768 resolution.

Figure 1-1. DemoQE128TFT LCD module installed on the DemoQE board

Figure 1-2. DemoQE128TFT 3.25” LCD module with touch screen installed onto the DemoQE board

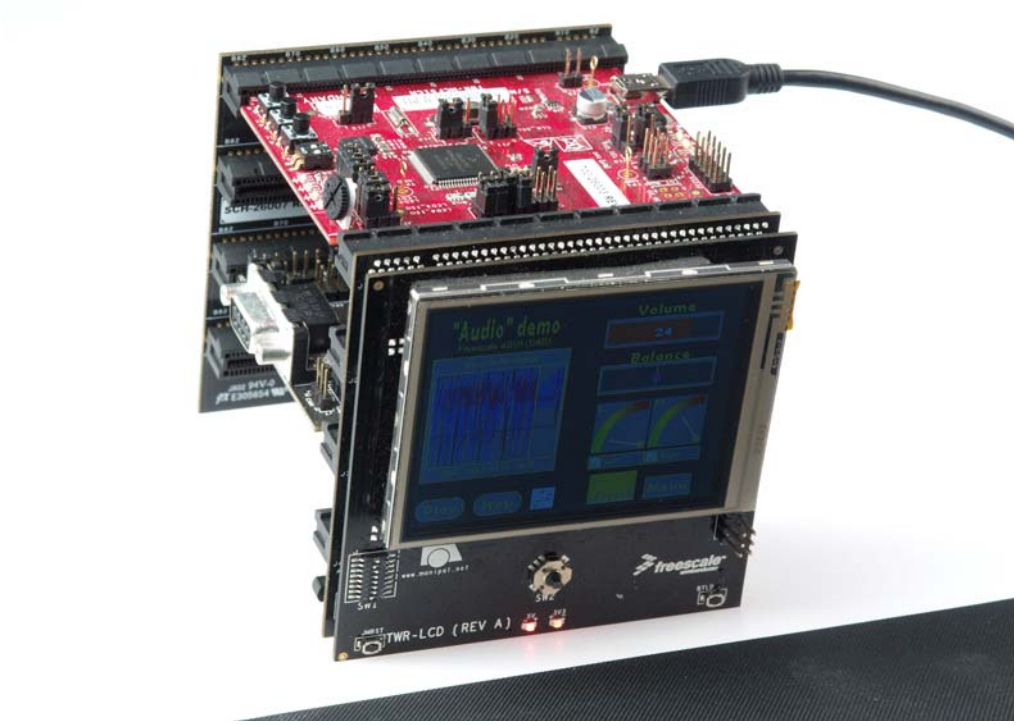


Figure 1-3. LCD module with touch screen installed onto the TOWER system

Figure 1-4. MCF52277 demo board with 8“ SVGA LCD

Figure 1-5. TWR-MCP5125 demo board with general digital display

Chapter 2
Description

2.1 Features

A solution for the LCD is needed in cases of control, measure, and diagnostics of some systems or devices with human-machine interface, where the operator can actively change conditions and requirements.

The eGUI/D4D is capable of generating the user menu, graphics, pictures, text, and display them on the LCD module. It allows interacting with all objects, dynamically changing, adding, or removing them. It also can read and write their status or current value. The D4D also fully supports touch screen capabilities of the LCD displays.

- Supports graphical color LCD displays of various sizes
- Small RAM (volatile) memory footprint
- Multiple platform support
- Object style of driver
- Smart support-screen-oriented structure of the user code
- Custom screen sizes, position, and a header like window
- Objects:
  - Button
  - Check Box / User handled Radio Button
  - Gauge
  - Icon
  - Label
  - Menu
  - Picture
  - Slider
  - Graph
  - Scroll Bar
  - Console
  - Text Box
- Touch screen support
- Multiple font support
- Buffer for input keys

2.2 Structure of Project with eGUI/D4D

Figure 2-1 shows the position of the D4D in the whole project. It is placed between low-level drivers of the LCD and the user application.

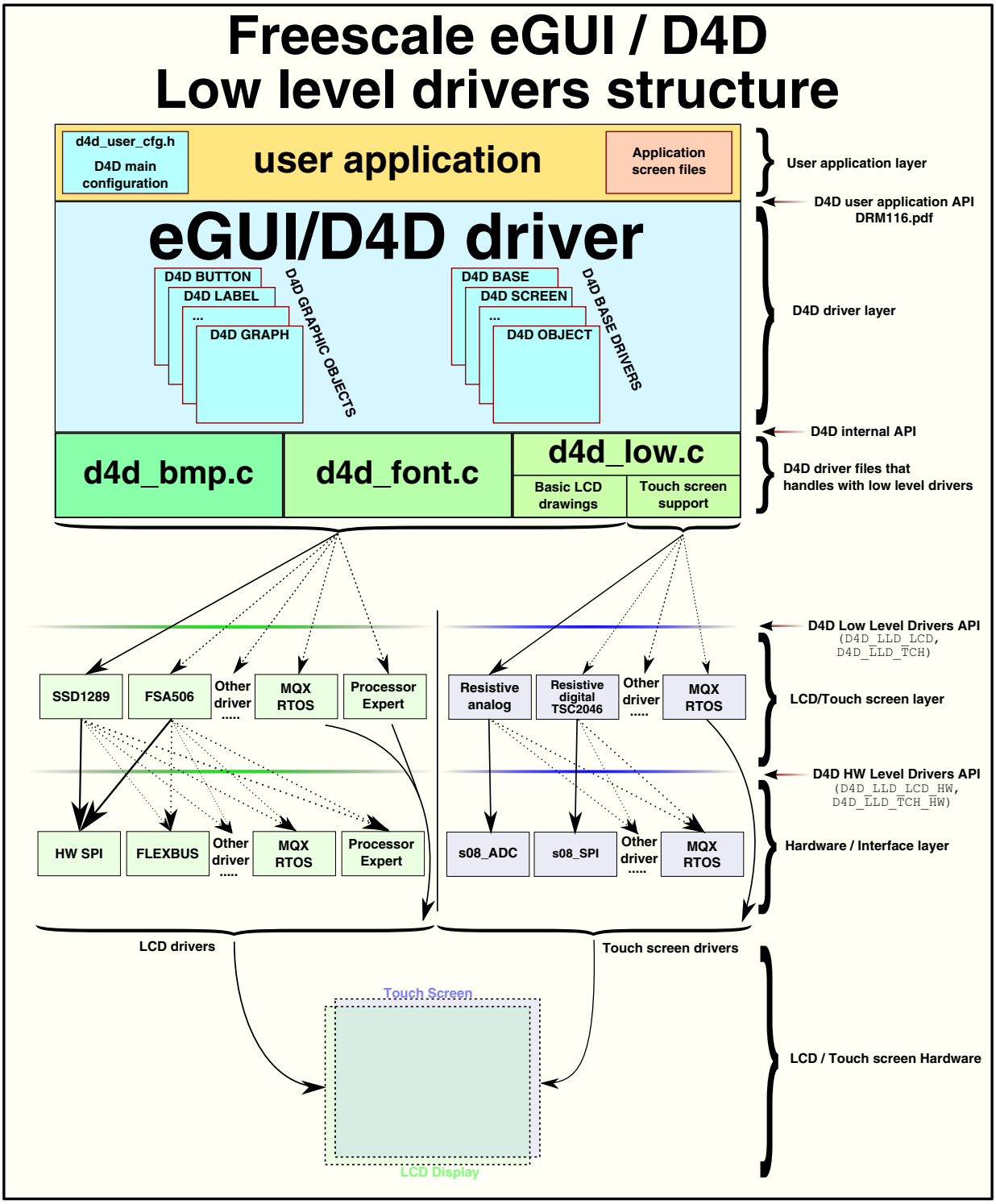


Figure 2-1. Freescale eGUI/D4D block diagram

2.3 File Structure

In Figure 2-2 you can find the file structure of eGUI/D4D, created from five types of files:

- LCD low-level driver
- LCD D4D high-level driver
- LCD D4D user configuration
- User application files
- User application D4D dependent file (screen files)

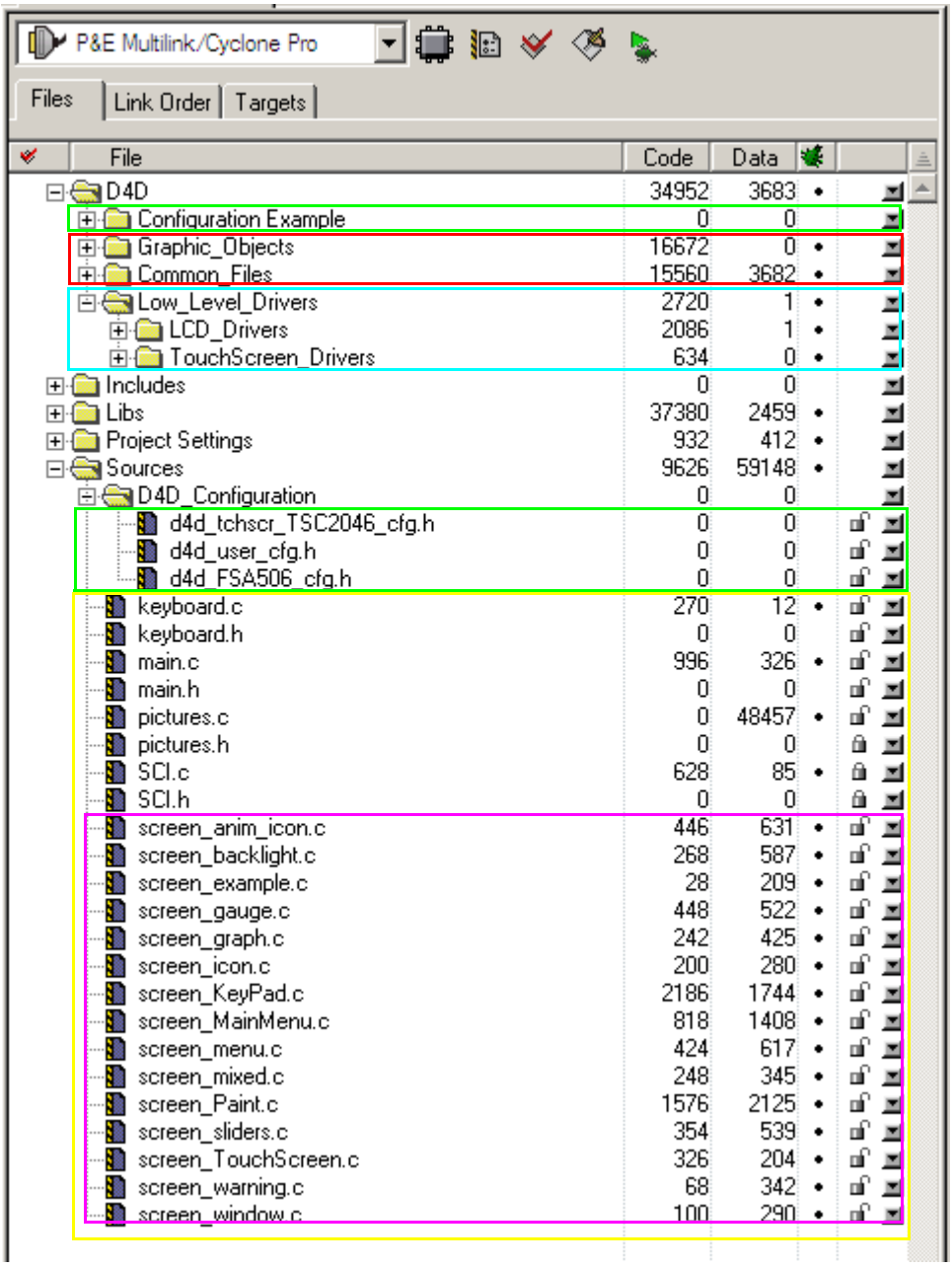


Figure 2-2. eGUI/D4D file structure in CodeWarrior project

2.3.1 eGUI/D4D Driver Files

All driver files are divided into four groups in the driver and configuration header files that the user has to add to their own application.

2.3.1.1 Common Files

The common files group contains all general files that are used by the entire driver and most of the user functions.

- d4d.h
  - Implements all header files
  - Provides all available function prototypes to the user
- d4d_private.h
  - Provides all available private function prototypes to the driver for internal use
- d4d_types.h
  - Provides common type prototypes of the driver
- d4d_lldapi.h
  - Provides API interface for the D4D low-level driver system
- d4d_base.h / d4d_base.c
  - Provides general C. Contains the core functions to run the whole driver.
- d4d_font.h / d4d_font.c
  - Provides functions and types for working with fonts
- d4d_low.h / d4d_low.c
  - These files provide all basic drawing functions and touch screen control functions. These functions make an interface between the high-level driver function and low-level drivers. It directly uses low-level functions.
- d4d_bmp.h / d4d_bmp.c
  - Provides functions and types for working with bitmaps
- d4d_math.h / d4d_math.c
  - Provides mathematical functions needed in objects as a rule of three and goniometric functions
- d4d_scheme.h / d4d_scheme.c
  - Provides functions/prototypes to create, change, and manage the driver color schemes
- d4d_screen.h / d4d_screen.c
  - Provides a function to create, change, and manage screens
- d4d_object.h / d4d_object.c
  - Provides a function to create, change, and manage objects

2.3.1.2 Configuration Example

The configuration example (d4d_user_cfg.h.example) contains the template of the driver configuration file that has to be copied into the user application and renamed to d4d_user_cfg.h.

The group also contains configuration file examples for all supported low-level drivers (the examples for the low level driver are physically placed in the low level drivers folder). The use of these drivers is the same as for the main configuration file. Copy it into the user project and rename it to **h** suffix.

2.3.1.3 Graphic Objects

The graphic object group contains all high-level graphical objects.

- d4d_button.h / d4d_button.c
  - Provides functions and prototypes to create, change, and manage the button
- d4d_check_box.h / d4d_check_box.c
  - Provides functions and prototypes to create, change, and manage the check box or user-managed radio button object
- d4d_gauge.h / d4d_gauge.c
  - Provides functions and prototypes to create, change, and manage the gauge object
- d4d_graph.h / d4d_graph.c
  - Provides functions and prototypes to create, change, and manage the simple graph object
- d4d_icon.h / d4d_icon.c
  - Provides functions and prototypes to create, change, and manage the icon object
- d4d_label.h / d4d_label.c
  - Provides functions and prototypes to create, change, and manage the label object
- d4d_menu.h / d4d_menu.c
  - Provides functions and prototypes to create, change, and manage the menu object
- d4d_picture.h / d4d_picture.c
  - Provides functions and prototypes to create, change, and manage the picture object
- d4d_slider.h / d4d_slider.c
  - Provides functions and prototypes to create, change, and manage the slider object
- d4d_scroll_bar.h / d4d_scroll_bar.c
  - Provides functions and prototypes to create, change, and manage the scroll bar object
- d4d_console.h / d4d_console.c
  - Provides functions and prototypes to create, change, and manage the console object
- d4d_text_box.h / d4d_text_box.c
  - Provides functions and prototypes to create, change, and manage the text box object

2.3.1.4 Low-Level Drivers

The low-level driver groups contain the hardware-dependent low-level drivers for the LCD/TFT displays and touch screens. This group is divided into two sub-groups. These groups are also divided into another two groups that separate controller drivers and hardware interface drivers. The lower level driver file structures are shown in Figure 2-3. This organization of low level drivers allow various combinations of LCD and Touch screen drivers and hardware interfaces as shown in Figure 2-1

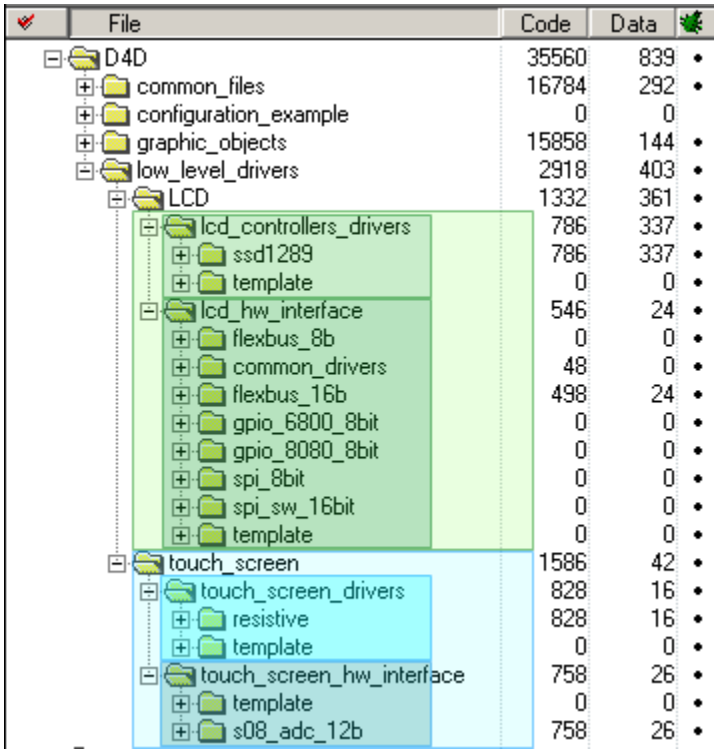


Figure 2-3. eGUI/D4D low level driver file structures

2.3.1.4.1 LCD Drivers

The LCD driver directory contains both layers designated for the LCD control.

LCD Controller Drivers

This group contains all supported LCD drivers.

- d4dlcd_ssd1289.h / d4dlcd_ssd1289.c
  - Provides functions and prototypes to control the LCD module with the SSD1289 TFT controller.
- d4dlcd_fsa506.h / d4dlcd_fsa506.c
  - Provides functions and prototypes to control the LCD module with the FSA506 TFT controller.
- d4dlcd_ls020.h / d4dlcd_ls020.c
  - Provides functions and prototypes to control the LS020 LCD module.
- d4dlcd_lgdp4531.h / d4dlcd_lgdp4531.c
  - Provides functions and prototypes to control the LCD module with the LGDP4531 TFT controller.
- d4dlcd_frame_buffer.h / d4dlcd_frame_buffer.c
  - Provides functions and prototypes to control the LCD module over the MCU/MPU peripheral.
- d4dlcd_template.h / d4dlcd_template.c
  - Template of the LCD low level driver, that must be used to create a new low level driver.

LCD Hardware Interface Drivers

This group contains all supported LCD driver hardware interfaces for the LCD controllers mentioned above.

- d4dlcdhw_flexbus_16b.h / d4dlcdhw_flexbus_16b.c
  - Provides functions and prototypes to communicate with the LCD controller through a 16-bit width 6800 parallel bus interface over flexbus.
- d4dlcdhw_flexbus_8b.h / d4dlcdhw_flexbus_8b.c
  - Provides functions and prototypes to communicate with the LCD controller through an 8-bit width 6800 parallel bus interface over flexbus.
- d4dlcdhw_gpio6800_8b.h / d4dlcdhw_gpio6800_8b.c
  - Provides functions and prototypes to communicate with the LCD controller through an 8-bit width 6800 parallel bus interface over gpio.
- d4dlcdhw_gpio8080_8b.h / d4dlcdhw_gpio8080_8b.c
  - Provides functions and prototypes to communicate with the LCD controller through an 8-bit width 8080 parallel bus interface over gpio.
- d4dlcdhw_gpio8080_byte_8b.h / d4dlcdhw_gpio8080_byte_8b.c
  - Provides functions and prototypes to communicate with the LCD controller through an 8-bit width 8080 parallel bus interface over gpio. This is a special derivative from a gpio8080 driver that sends only bytes instead of words. This access is needed for an FSA506 LCD controller.
- d4dlcdhw_s12_spi_16b.h / d4dlcdhw_s12_spi_16b.c
  - Provides functions and prototypes to communicate with the LCD controller through a 16-bit serial bus interface over an SPI. This driver is a derivative from the S12 MCU family.
- d4dlcdhw_spi_8b.h / d4dlcdhw_spi_8b.c
  - Provides functions and prototypes to communicate with the LCD controller through an 8-bit serial bus interface over spi. This driver is a derivative from the S08 MCU family.
- d4dlcdhw_spi_swc_8b.h / d4dlcdhw_spi_swc_8b.c
  - Provides functions and prototypes to communicate with software controlled LCD controller by 8-bit serial bus interface over spi. This driver is a derivative from the S08 MCU family.
- d4dlcdhw_spi_sw_16b.h / d4dlcdhw_spi_sw_16b.c
  - Provides functions and prototypes to communicate with the LCD controller through a 16-bit serial bus SPI interface over gpio.
- d4dlcd_template.h / d4dlcd_template.c
  - Template of the LCD low level hardware interface driver that must be used to create a new low level hardware interface driver.
- d4dlcdhw_dragonfire_lcdc.h / d4dlcdhw_dragonfire_lcdc.c
  - Provides functions and prototypes to operate with the MCF52277 LCDC peripheral.
- d4dlcdhw_mqx_mpc5125_diu.h / d4dlcdhw_mqx_mpc5125_diu.c
  - Provides functions and prototypes to operate with the MPC5125 DIU peripheral on the MQX (3.6 version or higher).

- d4dlcd_template_fb.h / d4dlcd_template_fb.c
  - LCD template of the low level hardware interface driver for the frame buffer that must be used to create a new frame buffer low level hardware interface driver.

2.3.1.4.2 Touch Screen Drivers

The LCD drivers directory contain both layers designated for the touch screen control.

Touch Screen Drivers

- d4dtch_resistive.h / d4dtch_resistive.c
  - Provides functions and prototypes to control the resistive touch screen module with analog interface
- d4dtch_TSC2046.h / d4dtch_TSC2046.c
  - Provides functions and prototypes to control the resistive touch screen module with a digital SPI interface provided by the TSC2046 controller
- d4dtch_mcf52277_asp.h / d4dtch_mcf52277_asp.c
  - Provides functions and prototypes to control the resistive touch screen module with an analog interface on the MCF52277 ASP peripherial. The driver is interrupt driven.
- d4dtch_template.h / d4dtch_template.c
  - Template of the touch screen low level driver that must be used to create a new low level driver

Touch Screen Hardware Interface Drivers

This group contains all supported touch screen driver hardware interfaces for the touch screen hardware mentioned above.

- d4dtchhw_s08_adc.h / d4dtchhw_s08_adc.c
  - Provides a function to run a resistive method of reading the touch screen using a gpio and ADC peripherial.
- d4dtchhw_s12_adc.h / d4dtchhw_s12_adc.c
  - Provides a funtion to run the resistive method of reading the touch screen using a gpio and ADC peripherial.This driver is a derivative from the S12 MCU family.
- d4dtchhw_mcf52259_adc.h / d4dtchhw_mcf52259_adc.c
  - Provides a funtion to run the resistive method of reading the touch screen using the gpio and ADC peripherial.This driver is derivative from the MCF52259 MCU.
- d4dtchhw_template.h / d4dtchhw_template.c
  - Template of the touch screen low level hardware interface driver that must be used to create a new low level hardware interface driver.

2.3.2 eGUI/D4D User Configuration File

This file is used to modify all default values and settings of the D4D. Here the behavior of the driver and its objects can be modified. The default visual aspect of the individual objects can also be modified.

- d4d_user_cfg.h

- This file is used to change default settings of the driver and all objects
- File has to be created in the user source code directory. The template of this file is placed in the driver folder, subfolder Configuration Example.

NOTE

All these files need to run the eGUI/D4D correctly in your project, including the d4d_user_cfg.h. If you do not use some functions you do not need to care about them, they will not be linked into your application and will not consume any memory or any other resources.

2.3.3 User Application Files

User application files are dependent on the target application, there are a few recommendations on how to use this driver and how to design source code for individual screens.

Recommendation:

- On start the user code void D4D_Init(D4D_SCREEN* pInitScreen) has to be called.
- In the main never-ending loop void D4D_Poll(void) has to be periodically called.
- When the keys were changed (for example in the keyboard interrupt) void D4D_NewKeyEvent(D4D_KEY_SCANCODE scanCode) has to be called with the new state of keys.
- In case the touch screen is used (for example in an interrupt routine from the timer peripheral) void D4D_CheckTouchScreen(void) has to be called periodically to check the state of the touch screen.
- In case the time events are used (for example in an interrupt routine from the timer peripheral) voidD4D_TimeTickPut(void) has to be called periodically, to provide time tick information into the driver.

2.3.3.1 User Application eGUI/D4D Dependent Files

- screen_xxxxxx.c
  - These files must contain all the code for individual screens as functions and data. Create a new file for each screen for better lucidity of the source code.

2.4 eGUI/D4D Data Memory Use

The driver consumes some flash memory for its code and static data structures. Here is an explanation of the memory usage for screens and objects. Each screen contains its own configuration information, its runtime data, and a list of all the objects that are present on it. Each object is put together from a general object representation, object specific data, and some objects also have run-time object data.

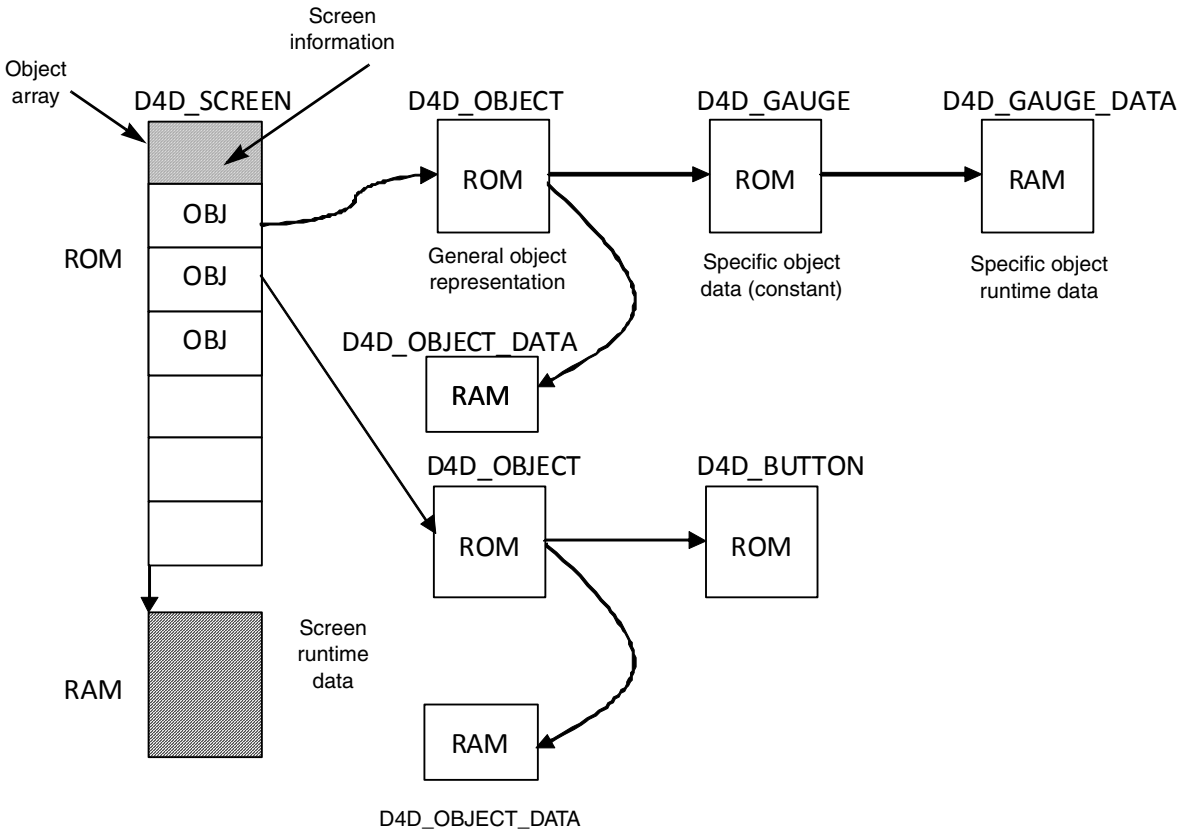


Figure 2-4. Memory map for screens and objects

Chapter 3
Driver API

3.1 Introduction

The following section describes the application programmers interface (API) on both sides of the D4D (user application and low-level driver).

3.2 eGUI/D4D to User Application API

3.2.1 Description

The description provides basic functions for creating, changing, handling objects and screens, and for managing their interactions.

3.2.2 General Types

This section describes all eGUI/D4D defined types that the user application can use for its application and for interaction with the driver.

Table 3-1. eGUI/D4D public Types

Types Name	Description	Alterable by cfg file
D4D_BOOL	The boolean type definition used in D4D. The size of this type depend on used MCU/MPU family from 8-bit to 32-bit to achieve using native variables.	No
D4D_COOR	Coordination data type, this must be larger than the resolution of the longer side of the LCD. The default value is an unsigned 8-bit integer.	Yes

D4D_POINT	This type is created as a structure from the two D4D_COOR types:
	`typedef struct D4D_POINT {`
	`  D4D_COOR x;`
	`  D4D_COOR y;`
	`} D4D_POINT;`
		No

D4D_SIZE	This type is created as structure of two D4D_COOR types. It is used to specify size of any object / elementary object in D4D.
	`typedef struct {`
	`  D4D_COOR cx;`
	`  D4D_COOR cy;`
	`} D4D_SIZE;`
		No

D4D_INDEX	Type definition for indexes used by the D4D. Unsigned 16-bit integer.	No
D4D_INDEX_DELTA	Type definition for signed indexes used by the D4D. Signed 16-bit integer.	No

Table 3-1. eGUI/D4D public Types (continued)

D4D_LINETYPE	Line type definition	No

D4D_BMP	Bitmap type definition, contains a pointer to a bitmap data array and color palette data array (could be NULL)
	`typedef struct {`
	`  const void *pData;`
	`  const void *pParam;`
	`} D4D_BMP;`
		No

D4D_CHAR	Type definition for chars used by the D4D. 8-bit integer.	No
D4D_COLOR	Type definition for colors used by the D4D. Unsigned 16-bit integer.	No
D4D_KEY_SCANCODE	Type definition of the scan code variable used for keys managed for the whole driver.	No
D4D_KEYS	Type definition of keys variable. Default value is unsigned 8-bit integer.	Yes

D4D_OBJECT_TYPE	Enum used to recognize the object created in the application for various cases.
	`typedef enum`
	`{`
	`D4D_OBJECT_BUTTON,`
	`D4D_OBJECT_CHECKBOX,`
	`D4D_OBJECT_GAUGE,`
	`D4D_OBJECT_GRAPH,`
	`D4D_OBJECT_ICON,`
	`D4D_OBJECT_LABEL,`
	`D4D_OBJECT_MENU,`
	`D4D_OBJECT_PICTURE,`
	`D4D_OBJECT_SLIDER`
	`D4D_OBJECT_CONSOLE,`
	`D4D_OBJECT_SCROLLBAR,`
	`D4D_OBJECT_TEXTBOX,`
	`D4D_OBJECT_UNKNOWN`
	`}D4D_OBJECT_TYPE;`
		No

D4D_OBJECT_SYS_FUNCTION	Internal type definition used to store object system function pointers.
	`typedef struct D4D_OBJECT_SYS_FUNCTION_S`
	`{`
	`  D4D_OBJECT_TYPE type;`
	`  void (*OnSysMessage)(struct D4D_MESSAGE_S* pMsg);`
	`  Byte (*CheckCoordinates)(struct D4D_OBJECT_S* pObj, D4D_POINT point);`
	`  struct D4D_TXTBUFF_S* (*GetTextBuffer)(struct D4D_OBJECT_S* pObj);`
	`}D4D_OBJECT_SYS_FUNCTION;`
		No

Table 3-1. eGUI/D4D public Types (continued)

D4D_OBJECT	Type definition of the graphic object.
	`typedef struct D4D_OBJECT_S`
	`{`
	`  const void* pParam;`
	`  Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);`
	`  D4D_OBJECT_SYS_FUNCTION* pObjFunc;`
	`  void *userPointer;`
	`  D4D_OBJECT_INITFLAGS initFlags;`
	`  D4D_OBJECT_FLAGS* flags;`
	`  struct D4D_CLR_SCHEME_S* clrScheme;`
	`  struct D4D_SCREEN_S** pScreen;`
	`} D4D_OBJECT;`
		No

D4D_SCREEN	Type definition of the screen. Screen structure contains a pointer to an array of all defined screen objects and a pointer of mandatory user callback functions.
	`typedef struct D4D_SCREEN_S`
	`{`
	`  // Object list`
	`  const D4D_OBJECT* const * pObjects; //`
	`NULL-terminated array of objects (may lay in ROM)`
	
	`  // Event handlers`
	`  void (*OnInit)(void); // One-time initialization`
	`  void (*OnMain)(void); // Main screen handler`
	`function`
	`  void (*OnActivate)(void); // Called before screen`
	`activation`
	`  void (*OnDeactivate)(void); // Called before`
	`deactivating`
	`  Byte (*OnObjectMsg)(struct D4D_MESSAGE_S* pMsg);`
	`// Called before object receives the message`
	
	`  // Screen advanced properties`
	`  D4D_POINT position;`
	`  D4D_SIZE size;`
	`  struct D4D_TXTBUFF_S textBuff;`
	`  const D4D_BMP* pIcon;`
	`  D4D_SCREEN_FLAGS flags;`
	`  struct D4D_CLR_SCHEME_S* clrScheme;`
	
	`  // Pointer to runtime data`
	`  D4D_SCREEN_DATA* pData;`
	
	`} D4D_SCREEN;`
		No

Table 3-1. eGUI/D4D public Types (continued)

D4D_OBJECT_DRAWFLAGS	Type definition of volatile object flags. These flags are available for the user application with D4D_MESSAGE with index D4D_MSG_DRAW.
	`typedef union {`
	`  D4D_BIT_FIELD all;`
	`  struct{`
	`  unsigned bVisible :1; // Visible at all`
	`  unsigned bEnabled :1; // Enabled or disabled option`
	`  unsigned bTabStop :1; // Can be focused`
	`  unsigned bTouchEnable :1; // Object is enabled for touch screen`
	`  unsigned bFastTouch :1; // Only one touch for action`
	`  unsigned bRedraw :1; // Needs to redraw`
	`  unsigned bRedrawC :1; // Use complete redraw`
	`  unsigned bInitDone :1; // Object initialization done`
	`  } bits;`
	`} D4D_OBJECT_FLAGS;`
		No

D4D_MSGID	Type definition of the list of all the possible D4D_MESSAGE indexes.
	`typedef enum`
	`{`
	`  D4D_MSG_DRAW, // Parameter wprm_DRAW`
	`  D4D_MSG_DRAWDONE, // Parameter wprm_DRAW`
	`  D4D_MSG_KEYUP, // Parameter wprm_KEYS`
	`  D4D_MSG_KEYDOWN, // Parameter wprm_KEYS`
	`  D4D_MSG_SETFOCUS, // No parameter`
	`  D4D_MSG_KILLFOCUS, // No parameter`
	`  D4D_MSG_SETCAPTURE, // No parameter`
	`  D4D_MSG_KILLCAPTURE, // No parameter`
	`  D4D_MSG_ONINIT, // No parameter`
	`  D4D_MSG_TOUCHED, // No parameter`
	`  D4D_MSG_TOUCH_AUTO, // No parameter`
	`  D4D_MSG_UNTOUCHED // No parameter`
	`  D4D_MSG_TIMETICK // No parameter`
	`} D4D_MSGID;`
		IIII

D4D_MESSAGE	Type definition of the system message. The message contains information of the destination screen and object. Message type (nMsgId) and possible data depends on the message type.
	`typedef struct D4D_MESSAGE{`
	`  D4D_OBJECT* pObject;`
	`  D4D_SCREEN* pScreen;`
	`  D4D_MSGID nMsgId;`
	`  union {`
	`  D4D_OBJECT_DRAWFLAGS draw;`
	`D4D_KEY_SCANCODE keys;`
	`  } prm;`
	`} D4D_MESSAGE;`
		No

D4D_FONT	Type definition of the font index. Default value is an unsigned 8-bit integer.	No

Table 3-1. eGUI/D4D public Types (continued)

D4D_FONT_TYPE	The definition of the font contains the font index, pointer on font data, and font scale.
	`typedef struct`
	`{`
	`  const D4D_FONT  ix_font;`
	`  const D4D_FONT_DESCRIPTOR* pFontDescriptor;`
	`  const D4D_FONT_SIZES scale;`
	`const D4D_FONT_SIZES charSpacing;`
	`const char* fileName;`
	`}D4D_FONT_TYPE;`
		No

D4D_FONT_DESCRIPTOR	The structure that contains all the necessary information about stored font, data, and all font sizes.
	`typedef struct{`
	`  D4D_FONT_REV revision; // Font descriptor version`
	`  D4D_FONT_FLAGS flags; // Linear or non linear,etc`
	`  D4D_FONT_IX startChar; // Start char of table used`
	`  D4D_FONT_IX charCnt; // End char of the table used`
	`  D4D_FONT_IX charDefault; // Default index of char`
	`D4D_FONT_SIZE charSize; // Font size`
	`D4D_FONT_SIZE charBmpSize; // Size of the font bitmpap`
	`D4D_FONT_SIZE charBaseLine; // Baseline offset`
	`  D4D_FONT_SIZES charFullSize; // Size of the biggest char x or y`
	`  D4D_FONT_PACK pack; // Packing condition of the bitmaps`
	`  D4D_FONT_IX *pIxTable; // Index table`
	`D4D_FONT_OFFSET *pOffTable; // Offset table`
	`D4D_FONT_SIZE *pSizeTable; // Size table const`
	`D4D_FONT_DATA *pFontData;// Bitmap or font data array`
	`}D4D_FONT_DESCRIPTOR;`
		No

D4D_FONT_DATA	Type definition of the font data pointer. The default value is pointer on an unsigned 8-bit integer.	No

D4D_FONT_SIZES	The structure of the font size in both directions —width and height.
	`typedef struct{`
	`FONT_SIZE width;`
	`FONT_SIZE height;`
	`}FONT_SIZES;`
		No

D4D_FONT_SIZE	Type definition font size variable. Default value is an unsigned 8-bit integer.	No
D4D_FONT_REV	The font descriptor type revision. Unsigned 8-bit integer.	No
D4D_FONT_FLAGS	Type definition of the stored font. Unsigned 8-bit integer.	No
D4D_FONT_IX	Type definition of the char index. Unsigned 8-bit integer.	No
D4D_FONT_PACK	Type definition of the stored font bitmap. Unsigned 8-bit integer.	No

Table 3-1. eGUI/D4D public Types (continued)

D4D_FONT_OFFSET	The Internal type used for the stored offset of the char data in the whole font data area. Unsigned 16-bit integer.	No

D4D_ORIENTATION	Diplay orientation enum type.
	`typedef enum`
	`{`
	`  D4D_ORIENT_PORTRAIT,`
	`  D4D_ORIENT_PORTRAIT180,`
	`  D4D_ORIENT_LANDSCAPE,`
	`  D4D_ORIENT_LANDSCAPE180`
	`} D4D_ORIENTATION;`
		No

D4D_QUADRANT	The enumeration type definition of circle quadrant. It is used in rounded corners drawing functions parameters.
	`typedef enum`
	`{`
	`D4D_QUAD_1,`
	`D4D_QUAD_2,`
	`D4D_QUAD_3,`
	`D4D_QUAD_4`
	`}D4D_QUADRANT;`
		No

D4D_TOUCHSCRENN_CALIB	This structure contains all calibration constants to the recalculated input RAW value of the touch screen to output value in pixels.
	`typedef struct {`
	`  unsigned char LCD_ScreenCalibrated;`
	`  unsigned int LCD_TouchScreenXoffset;`
	`  unsigned int LCD_TouchScreenYoffset;`
	`  unsigned int LCD_TouchScreenXBitsPerPixelx10;`
	`  unsigned int LCD_TouchScreenYBitsPerPixelx10;`
	`}D4D_TOUCHSCREEN_CALIB;`
		No

3.2.2.1 eGUI/D4D Public General Defines

The driver provides a few general public defines that are recommended for use with the D4D API.

Boolean values are used in many cases in the driver:

- D4D_TRUE
- D4D_FALSE

The return values from the user application driver system message callback is used to skip the current message flow to the driver objects:

- D4D_MSG_NOSKIP
- D4D_MSG_SKIP

3.2.2.2 eGUI/D4D Public Predefined Color

The driver provides a few predefined basic colors in RGB 5-6-5 format.

These colors can be used with D4D_COLOR type:

- D4D_COLOR_DARK_BLUE
- D4D_COLOR_BRIGHT_BLUE
- D4D_COLOR_BLUE
- D4D_COLOR_BRIGHT_YELLOW
- D4D_COLOR_YELLOW
- D4D_COLOR_ORANGE
- D4D_COLOR_BRIGHT_RED
- D4D_COLOR_RED
- D4D_COLOR_DARK_RED
- D4D_COLOR_BRIGHT_GREEN
- D4D_COLOR_GREEN
- D4D_COLOR_DARK_GREEN
- D4D_COLOR_WHITE
- D4D_COLOR_LIGHT_GREY
- D4D_COLOR_GREY
- D4D_COLOR_BLACK

3.2.2.3 eGUI/D4D Public Predefined Line Types

The driver provides two predefined line types. These types can be used with the D4D_LINE type:

- D4D_LINE_THIN
- D4D_LINE_THICK

3.2.2.4 eGUI/D4D Public Predefined Key Types

The driver provides a few predefined constants that modify the behavior of the key input module. All of these constants can be modified in the user configuration file.

The size of the buffer for input key events:

- D4D_KEYS_BUFF_LENGTH (default value: 4)

These are the system key scancodes of the supported action keys. The default values are set to meet scancodes from the original PC XT specification. The scan code is set up from the key 7-bit scancode value and the MSB bit that manages information about the state of the key pushed or released:

- D4D_KEY_SCANCODE_UP (default value: 0x51)
- D4D_KEY_SCANCODE_DOWN (default value: 0x50)
- D4D_KEY_SCANCODE_LEFT (default value: 0x4B)

- D4D_KEY_SCANCODE_RIGHT (default value: 0x4D)
- D4D_KEY_SCANCODE_ENTER (default value: 0x1C)
- D4D_KEY_SCANCODE_ESC (default value: 0x01)

For back compatibility the old style of key definitions by masks are kept.

The system key masks. This mask specifies the individual bits in D4D_KEYS type:

- D4D_KEY_UP (default value: 0x01)
- D4D_KEY_DOWN (default value: 0x02)
- D4D_KEY_LEFT (default value: 0x04)
- D4D_KEY_RIGHT (default value: 0x08)
- D4D_KEY_ENTER (default value: 0x10)
- D4D_KEY_ESC (default value: 0x20)

3.2.2.5 eGUI/D4D Public Predefined Init General Object Flags

Each object in the D4D driver has some init flags that indicate the behavior of initialized objects. The init flags are divided into two groups, first is the system group and all the objects have the same flags. Second group of flags are object dependent.

The list of system init flags:

- D4D_OBJECT_F_VISIBLE—Object after initialization is visible on the screen
- D4D_OBJECT_F_ENABLED—Object after initialization is enabled
- D4D_OBJECT_F_TABSTOP—Object can be focused
- D4D_OBJECT_F_TOUCHENABLE—Object has enabled touch screen capability
- D4D_OBJECT_F_FASTTOUCH—Object has enabled fast touch screen capability. This option supports only a few objects (button, check box, and label).
- D4D_OBJECT_F_TRANSP_TEXT—The main text of an object is drawn as transparent without a background color
- D4D_OBJECT_F_FOCUSRECT—Object has an outlined rectangle

3.2.2.6 eGUI/D4D General Instantiation Macros

D4D_DECLARE_BMP(name, pbmp, ppal);

Input parameters:

- name—This is the name of the declared bitmap
- pbmp—Pointer to a bitmap array
- ppal—Pointer to a palette array. It can be NULL when the bitmap coding does not use a palette.

Description—This instantiation is used to declare a new bitmap array. This declaration is furthermore used in all picture and icon declarations for all D4D objects.

3.2.3 General Driver Calls

There are a few general functions that are key to initialize and use the D4D.

void D4D_Init(D4D_SCREEN* pInitScreen);

Input parameters:

- pInitScreen—Pointer to the initialization screen of the user application

Output parameters—Result of an init operation. D4D_TRUE for success and D4D_FALSE for failed initialization.

Description—This function initializes the internal variables of the D4D driver and initializes the low-level driver.

A call to this function must occur before any call to the D4D driver API function.

void D4D_Poll(void);

Input parameters—NA

Output parameters—NA

Description—This is the main function of the D4D that manages all screen and object refreshes, key inputs, and touch screen events. It is recommended to place a call of this function into a main loop of the user application. It can be placed to a part of the main loop with the lowest priority.

A call of this function must be placed into the main loop of user application to run the D4D correctly.

void D4D_NewKeyEvent(D4D_KEY_SCANCODE scanCode);

Input parameters:

- scanCode(in)—scanCode of key event

Output parameters—NA

Description—This function is used to notify the D4D that a new input key event occurred. This function adds a new key event into the internal buffer of the input events. This function can be called from the interrupt routines.

void D4D_KeysChanged(D4D_KEYS keys);

Input parameters:

- keys (in)—Mask of an actual set of inputs (keys)

Output parameters—NA

Description—This function is used to notify the D4D that the input keys (or general input device) have been changed. Each call of this function adds a new state of keys into the internal buffer of the input events. This function can be called from the interrupt routines.

NOTE

This function is only for backward compatibility with a previous version of the D4D. This function can support only D4D system keys. Due to the switch, the complete project to run with key scancodes, the main key event input function is D4D_NewKeyEvent.

3.2.4 Touch Screen Driver Calls

The following functions are required for driver touch screen capability.

void D4D_CheckTouchpad(void);

Input parameters—NA

Output parameters—NA

Description—This function runs internal check routines to get a new status of the touch screen untouched, touched, and coordination. This function can be called from the interrupt routines.

D4D_TOUCHSCREEN_CALIB D4D_GetTouchScreenCalibration(void);

Input parameters—NA

Output parameters—Structure with touch screen calibration data

Description—The function returns the touch screen calibration data.

void D4D_SetTouchScreenCalibration(D4D_TOUCHSCREEN_CALIB newCalib);

Input parameters—Structure with touch screen calibration data

Output parameters—NA

Description—The function sets new touch screen calibration data.

void D4D_CalibrateTouchScreen(void);

Input parameters—NA

Output parameters—NA

Description—The function clears the current calibration data and runs the calibration process. After calibration finishes the function invalidates the last screen and redraws it.

D4D_POINT D4D_GetTouchScreenCoordinates(void);

Input parameters—NA

Output parameters—Structure of a point

Description—The function returns the last touched point on the client screen.

void D4D_PutTouchScreen(D4D_BOOL touched, D4D_COOR x, D4D_COOR y);

Input parameters:

touched—Boolean value of touch screen status

x—Coordination of touch in axis X

y—Coordination of touch in axis Y

Output parameters—NA

Description—The function puts new “touch” information into the driver. It is used for projects that do not use native touch screen low level drivers to get touch screen information.

void D4D_PutRawTouchScreen(D4D_BOOL touched, D4D_COOR x, D4D_COOR y);

Input parameters

touched—Boolean value of touch screen status

x—Coordination of touch in axis X

y—Coordination of touch in axis Y

Output parameters—NA

Description—The function puts new “touch” information into the driver. It is used for projects that do not use native touch screen low level drivers to get touch screen information. The input values are not calibrated.

3.2.5 eGUI/D4D Color Schemes

The whole driver objects and screens use the color schemes. The driver also contains one default color scheme used as default for all objects that do not have any other color scheme set in the declaration.

The eGUI/D4D color schemes have a few basic but useful advantages:

- Basic object and screen colors over the whole driver usage
- Allows using a different color scheme for each unique object and screen
- Driver allows the run-time change of the default color scheme (for example useful for day and night colors of application)

3.2.5.1 D4D_SCHEME Instantiation Macros

The color scheme has a simple instantiation macro that specifies all colors for screens, common object colors, and object dependent colors. This instantiation macro creates a color scheme in the non-volatile memory (ROM).

D4D_DECLARE_CLR_SCHEME(name, scrDesktop, scrHeader, scrFore, scrForeDis, scrExitBtnFore, scrExitBtnBckg, objBckg, objBckgDis, objBckgFocus, objBckgCapture, objFore, objForeDis, objForeFocus, objForeCapture, gaugHub, gaugPointer, sldrBarBckg, sldrBar, sldrBarStart, sldrBarEnd, iconBckg)

Input parameters:

- name—Name of the color scheme
- scrDesktop—Used as a background and desktop color of screen
- scrHeader—Used as a color of the screen header

- scrFore—Used as a forecolor of the screen (example: title text and outline)
- scrForeDis—Used as a forecolor of the screen in disable state
- scrExitBtnFore—Used as a forecolor of the screen exit button
- scrExitBtnBckg—Used as a forecolor of the screen exit button in disable state
- objBckg—Used as a background color of objects
- objBckgDis—Used as a background color of objects in disable state
- objBckgFocus—Used as a background color of objects in focus state
- objBckgCapture—Used as a background color of objects in capture state
- objFore—Used as a forecolor of objects
- objForeDis—Used as a forecolor of objects in disable state
- objForeFocus—Used as a forecolor of objects in focus state
- objForeCapture—Used as a forecolor of objects in capture state
- gaugHub—Used as a color for hub in gauge object
- gaugPointer—Used as a color for pointer in gauge object
- sldrBarBckg—Used as a background color for bar in slider object
- sldrBar—Used as a forecolor for bar in slider object
- sldrBarStart—Used as a start color for the bar in the slider object, when the auto color of the bar option is enabled
- sldrBarEnd—Used as an end color for a bar in slider object, when the auto color of the bar option is enabled
- iconBckg—Used as a background color for icon in check box object

Description—This is a full definition macro that allows setting of all colors in D4D color scheme.

3.2.5.2 D4D_SCHEME Type

The color scheme type is divided into three main parts:

- Screen colors
- Common object colors
- Object dependent colors

The screen and object dependent colors are placed in their own types to obtain a better organization inside of this structure.The common object colors are placed directly inside of this main color scheme structure.

Prototype of the D4D_SCHEME structure:

```
typedef struct D4D_CLR_SCHEME_S {
  D4D_CLR_SCHEME_SCR screen;
  D4D_COLOR bckg;
  D4D_COLOR bckgDis;
  D4D_COLOR bckgFocus;
  D4D_COLOR bckgCapture;
  D4D_COLOR fore;
  D4D_COLOR foreDis;
  D4D_COLOR foreFocus;
```

```
  D4D_COLOR foreCapture;
  D4D_CLR_SCHEME_OBJ objectDepend;
} D4D_CLR_SCHEME;
```

Description—This is the main structure of the color scheme in the D4D. It contains all the necessary colors to run the whole driver under the common color scheme. The screen colors are placed in D4D_CLR_SCHEME_SCR structure and object dependent colors are placed in D4D_CLR_SCHEME_OBJ structure.

Prototype of the D4D_SCR_CLR_SCHEME structure:

```
typedef struct {
  D4D_COLOR desktop;
  D4D_COLOR header;
  D4D_COLOR fore;
  D4D_COLOR foreDis;
  D4D_COLOR exitBtnFore;
  D4D_COLOR exitBtnBckg;
} D4D_SCR_CLR_SCHEME;
```

Description—This is the structure of the color scheme for screens in the D4D. It contains all the necessary colors to draw a screen under the common color scheme.

Prototype of D4D_CLR_SCHEME_OBJ:

```
typedef struct {
  D4D_CLR_SCHEME_GAUG   gauge;
  D4D_CLR_SCHEME_SLDR   slider;
  D4D_CLR_SCHEME_CHECKB checkBox;
} D4D_CLR_SCHEME_OBJ;
```

Description—This structure contains all object dependent colors of the color scheme in the D4D. It is divided into the small objects that describe structures containing individual colors.

Prototype of D4D_CLR_SCHEME_GAUG:

```
typedef struct{
  D4D_COLOR hub;
  D4D_COLOR pointer;
}D4D_CLR_SCHEME_GAUG;
```

Description—This structure contains specific colors (non-standard) for the gauge object.

Prototype of D4D_CLR_SCHEME_SLDR:

```
typedef struct{
  D4D_COLOR barBckg;
  D4D_COLOR bar;
  D4D_COLOR barStart;
  D4D_COLOR barEnd;
}D4D_CLR_SCHEME_SLDR;
```

Description—This structure contains specific colors (non-standard) for a slider object.

Prototype of D4D_CLR_SCHEME_CHECKB:

```
typedef struct{
  D4D_COLOR iconBckg;
}D4D_CLR_SCHEME_CHECKB;
```

Description—This structure contains specific colors (non-standard) for checkbox object.

3.2.5.3 D4D_SCHEME Predefined Standard Constants

The color scheme contains a few predefined constants for colors that are used in the default object declarations. All of these constants can be modified in the user configuration file.

The default forecolor:

- D4D_COLOR_FORE_NORM (default value: D4D_COLOR_BLACK)

The default forecolor in disable state:

- D4D_COLOR_FORE_DISABLED (default value: D4D_COLOR_GREY)

The default forecolor in focus state:

- D4D_COLOR_FORE_FOCUS (default value: D4D_COLOR_BRIGHT_RED)

The default forecolor in capture state:

- D4D_COLOR_FORE_CAPTURE (default value: D4D_COLOR_BRIGHT_GREEN)

The default background color:

- D4D_COLOR_BCKG_NORM (default value: D4D_COLOR_WHITE)

The default background color in disable state:

- D4D_COLOR_BCKG_DISABLED (default value: D4D_COLOR_LIGHT_GREY)

The default background color in focus state:

- D4D_COLOR_BCKG_FOCUS (default value: D4D_COLOR_GREY)

The default background color in capture state:

- D4D_COLOR_BCKG_CAPTURE (default value: D4D_COLOR_GREEN)

3.2.5.4 D4D_SCHEME Functions

D4D_CLR_SCHEME* D4D_ObjectGetScheme(D4D_OBJECT * pObj);

Input parameters:

- pObj—Pointer to object

Output parameters—Pointer to a color scheme

Description—Function returns pointer to a color scheme used by the object specified in the parameter.

D4D_CLR_SCHEME* D4D_ScreenGetScheme(D4D_SCREEN * pScreen);

Input parameters:

- pScreen—Pointer to an object

Output parameters—Pointer to a color scheme

Description—Function returns pointer to a color scheme that is used by a screen specified in the parameter.

D4D_CLR_SCHEME* D4D_GetSchemeDefault(void);

Input parameter—NA

Output parameters—Pointer to a color scheme

Description—Function returns pointer to a current default color scheme.

void D4D_SetSchemeDefault(D4D_CLR_SCHEME* pScheme);

Input parameters—Pointer to a color scheme

Output parameters—NA

Description—Function sets a new default color scheme by the input parameter.

D4D_COLOR D4D_ObjectGetForeColor(D4D_OBJECT * pObj, D4D_OBJECT_DRAWFLAGS draw);

Input parameters:

- pObj—Pointer to an object
- draw—Draw structure (indicates in full details the state of object)

Output parameters—Forecolor of an object

Description—Function returns the forecolor of an object. This function returns forecolor in full a scale of possible colors (normal, disable, focus, and capture state).

D4D_COLOR D4D_ObjectGetBckgColor(D4D_OBJECT * pObj, D4D_OBJECT_DRAWFLAGS draw);

Input parameters:

- pObj—Pointer to an object
- draw—Draw structure (indicates in full details the state of object)

Output parameters—Background color of an object

Description—Function returns background color of an object. This function returns background color in a full scale of possible colors (normal, disable, focus, and capture state).

D4D_COLOR D4D_ObjectGetForeSimpleColor(D4D_OBJECT * pObj);

Input parameters:

- pObj—Pointer to an object

Output parameters—Forecolor of an object

Description—Function returns the forecolor of an object. This function returns forecolor in a restricted scale of possible colors (normal and disable colors). Disable color, when the object is disabled. Normal color for all rest states.

D4D_COLOR D4D_ObjectGetBckgSimpleColor(D4D_OBJECT * pObj);

Input parameters:

- pObj—Pointer to an object

Output parameters—Background color of an object

Description—Function returns background color of an object. This function returns background color in a restricted scale of possible colors (normal and disable colors). Disable color, when the object is disabled and the normal color for all the rest states.

3.2.6 eGUI/D4D Fonts

The eGUI/D4D supports using multiple fonts in each project in a defined font table that indicates what fonts will be used in the project and allows creating font derivatives with a scaling and spacing function.

The eGUI/D4D allows support to whatever font type as defined in the d4d_font.c file, because all objects (also screen) work only with font indexes that point to the eGUI/D4D font table created by the user.

eGUI/D4D font support features:

- Multiple font creation based on same font data base, using scaling and spacing function
- Scaling in both axes
- Line and char spacing
- MonoSpace and proportional font support
- Full and restricted font table support (linear and non-linear)
- Two types of index tables for restricted
- Various types of font bitmap type
- Underline, strike through, and transparent font drawing support

3.2.6.1 D4D_FONT Types

The eGUI/D4D font support contains a few types that helps manage work with fonts and text under a more flexible driver.

Prototype of the D4D_FONT_PROPERTIES structure:

```
typedef union
{
    D4D_BIT_FIELD all;

    struct
    {
        unsigned bUnderLine     :2;   // font underline
        unsigned bStrikeThrough :2;   // strike through the text
        unsigned bTransparent   :1;   // trasparent text flag
        unsigned bReserve0      :1;   // Reserved bit
        unsigned bReserve1      :1;   // Reserved bit
        unsigned bReserve2      :1;   // Reserved bit
    } bits;
} D4D_FONT_PROPERTIES;
```

Description—This is the structure that contains font drawing properties. The main container is a union that allows the user to access data via a variable or bitfield. The type contains information of underline, strike through, and transparent property of the font.

Prototype of the D4D_TEXT_PROPERTIES structure:

```
typedef union
{
    D4D_BIT_FIELD all;

    struct
    {
        unsigned bAlignHoriz     :2;   // String horizontal alignment in text box
        unsigned bAlignVertic    :2;   // String vertical alignment in text box
        unsigned bReserverd     :4;   // Reserved bits
    } bits;
} D4D_TEXT_PROPERTIES;
```

Description—This is a structure that contains text drawing properties. The main container is a union that allows the user to access data via a variable or bitfield. The type contains information about the whole text alignment used in D4D_DrawTextRect function.

3.2.6.2 D4D_FONT Standard Constants

The eGUI/D4D font support contains a few predefined constants for font and text properties that are used in default object declarations and can be used by the user application.

The font properties macros for underline property:

- D4D_FNT_PRTY_UNDERLINE_NONE—No underlined font—This macro is used for setting and checking the bitfield in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_UNDERLINE_LINE—Underlined font—This macro is used for setting and checking bitfield in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_UNDERLINE_DOT—Underlined font by dots—This macro is used for setting and checking the bitfield in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_UNDERLINE_MASK—This macro is used for setting and checking all underline bits by mask in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_UNDERLINE_NONE_MASK—No underlined font—This macro is used for setting and checking by mask all the items in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_UNDERLINE_LINE_MASK—Underlined font—This macro is used for setting and checking by mask all the items in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_UNDERLINE_DOT_MASK—Underlined font by dots—This macro is used for setting and checking by mask all the items in D4D_FONT_PROPERTIES.

The font properties macros for strike through property:

- D4D_FNT_PRTY_STRIKETHROUGH_NONE—No strike through font—This macro is used for setting and checking the bitfield in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_STRIKETHROUGH_SINGLE—Strike through font by one line—This macro is used for setting and checking the bitfield in D4D_FONT_PROPERTIES.

- D4D_FNT_PRTY_STRIKETHROUGH_DOUBLE—Strike through font by two lines—This macro is used for setting and checking the bitfield in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_STRIKETHROUGH_TRIPLE—Strike through font by three lines—This macro is used for setting and checking the bitfield in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_STRIKETHROUGH_MASK—This macro is used for setting and checking all strike through bits by mask in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_STRIKETHROUGH_NONE_MASK—No strike through font—This macro is used for setting and checking by mask all the items in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_STRIKETHROUGH_SINGLE_MASK—Strike through font by one line—This macro is used for setting and checking by mask all the items in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_STRIKETHROUGH_DOUBLE_MASK—Strike through font by two lines—This macro is used for setting and checking by mask all the items in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_STRIKETHROUGH_TRIPLE_MASK—Strike through font by three lines—This macro is used for setting and checking by mask all the items in D4D_FONT_PROPERTIES.

The font properties macros for transparent property:

- D4D_FNT_PRTY_TRANSPARENT_NO—Normal drawing of font—This macro is used for setting and checking the bitfield in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_TRANSPARENT_YES—Transparent (without background) drawing of font—This macro is used for setting and checking the bitfield in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_TRANSPARENT_MASK—This macro is used for setting and checking the transparent bit by mask in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_TRANSPARENT_NO_MASK—Normal drawing of font. This macro is used for setting and checking by mask all the items in D4D_FONT_PROPERTIES.
- D4D_FNT_PRTY_TRANSPARENT_YES_MASK—Transparent (without background) drawing of font—This macro is used for setting and checking by mask all the items in D4D_FONT_PROPERTIES.

The text properties macros for horizontal alignment property:

- D4D_TXT_PRTY_ALIGN_H_LEFT—Horizontal left alignment of text—This macro is used for setting and checking bitfield in D4D_TEXT_PROPERTIES.
- D4D_TXT_PRTY_ALIGN_H_RIGHT—Horizontal right alignment of text—This macro is used for setting and checking the bitfield in D4D_TEXT_PROPERTIES.
- D4D_TXT_PRTY_ALIGN_H_CENTER—Horizontal center alignment of text—This macro is used for setting and checking the bitfield in D4D_TEXT_PROPERTIES.
- D4D_TXT_PRTY_ALIGN_H_MASK—This macro is used for setting and checking all horizontal alignment bits by mask in D4D_TEXT_PROPERTIES.
- D4D_TXT_PRTY_ALIGN_H_LEFT_MASK—Horizontal left alignment of text—This macro is used for setting and checking by mask all the items in D4D_TEXT_PROPERTIES.

- D4D_TXT_PRTY_ALIGN_H_RIGHT_MASK—Horizontal right alignment of text—This macro is used for setting and checking by mask all the items in D4D_TEXT_PROPERTIES.
- D4D_TXT_PRTY_ALIGN_H_CENTER_MASK—Horizontal center alignment of text—This macro is used for setting and checking by mask all the items in D4D_TEXT_PROPERTIES.

The text properties macros for vertical alignment property:

- D4D_TXT_PRTY_ALIGN_V_TOP—Vertical top alignment of text—This macro is used for setting and checking bitfield in D4D_TEXT_PROPERTIES.
- D4D_TXT_PRTY_ALIGN_V_BOTTOM—Vertical bottom alignment of text—This macro is used for setting and checking bitfield in D4D_TEXT_PROPERTIES.
- D4D_TXT_PRTY_ALIGN_V_CENTER—Vertical center alignment of text—This macro is used for setting and checking bitfield in D4D_TEXT_PROPERTIES.
- D4D_TXT_PRTY_ALIGN_V_MASK—This macro is used for setting and checking all vertical alignment bits by mask in D4D_TEXT_PROPERTIES.
- D4D_TXT_PRTY_ALIGN_V_TOP_MASK—Vertical top alignment of text—This macro is used for setting and checking by mask all the items in D4D_TEXT_PROPERTIES.
- D4D_TXT_PRTY_ALIGN_V_BOTTOM_MASK—Vertical bottom alignment of text—This macro is used for setting and checking by mask all the items in D4D_TEXT_PROPERTIES.
- D4D_TXT_PRTY_ALIGN_V_CENTER_MASK—Vertical center alignment of text—This macro is used for setting and checking by mask all the items in D4D_TEXT_PROPERTIES.

3.2.6.3 D4D_FONT Instantiation Macros

The font definition macro is created from three individual parts that allow to indicate a complete font table with various counts of used fonts:

- Begin part—This part creates the head of fonts array.
- Add fonts part—This part allows to use multiple D4D_DECLARE_FONT macros to add all project fonts.
- End part—This part is used only to close the font array definition.

D4D_DECLARE_FONT_TABLE_BEGIN

Description—This is the only mandatory part of the font table definition that creates a header of font array in the D4D. It has to be placed before the declaration part of individual fonts.

D4D_DECLARE_FONT(fontId, font_descriptor, xScale, yScale, charSpace, lineSpace)

Input parameters:

- fontId—A unique number that is used in the whole application as an index of this font.
- font_descriptor—Pointer to the source data of the font, it has to point to the D4D_FONT_DESCRIPTOR structure.
- xScale—Scale of the used font in axis X
- yScale—Scale of the used font in axis Y
- charSpace—Extra pixels count between char—char spacing

- lineSpace—Extra pixels count between lines—line spacing

Description—This macro can be used multiple times to create a list of fonts used in the project.

D4D_DECLARE_FONT_TABLE_END

Description—This macro closes only the fonts array definition.

3.2.6.3.1 Example to Use D4D_FONT Instantiation macros

This is an example of one definition of the standard font table that takes source data from two font structures (Font1_5x8, and Font2_8x14) and multiplies these two source data into eight various fonts.

In this example fontId (first parameter) defines are used. These defines are placed in the main application header file to be shared by all user application files.

```
D4D_DECLARE_FONT_TABLE_BEGIN
D4D_DECLARE_FONT_TABLE(FONT_5x8_SMALL, Font1_5x8, 1, 1, 1, 1)
D4D_DECLARE_FONT_TABLE(FONT_5x8_HIGH, Font1_5x8, 1, 2, 1, 1)
D4D_DECLARE_FONT_TABLE(FONT_5x8_WIDE, Font1_5x8, 2, 1, 1, 1)
D4D_DECLARE_FONT_TABLE(FONT_5x8_BIG, Font1_5x8, 2, 2, 1, 1)
D4D_DECLARE_FONT_TABLE(FONT_8x14_SMALL, Font2_8x14, 1, 1, 1, 1)
D4D_DECLARE_FONT_TABLE(FONT_8x14_HIGH, Font2_8x14, 1, 2, 1, 1)
D4D_DECLARE_FONT_TABLE(FONT_8x14_WIDE, Font2_8x14, 2, 1, 1, 1)
D4D_DECLARE_FONT_TABLE(FONT_8x14_BIG, Font2_8x14, 2, 2, 1, 1)
D4D_DECLARE_FONT_TABLE_END
```

3.2.6.4 eGUI/D4D Fonts Functions

The eGUI/D4D contains a few functions that are used for working with fonts and its properties.

D4D_FONT_TYPE* D4D_GetFont(D4D_FONT ix);

Input parameters:

- ix—Font index

Output parameters—Pointer to font structure.

Description—Function returns the pointer on the font type structure defined by the input parameter ix. The function goes through the font table and looks for font with the same index managed by ix. If the font table does not contain font with correct index functions, it returns NULL.

D4D_FONT_SIZES D4D_GetFontSize(D4D_FONT ix);

Input parameters:

- ix—Font index

Output parameters—Font size structure

Description—Function returns the font size structure filled by sizes of the font defined by the input parameter ix.

D4D_FONT_SIZE D4D_GetFontHeight(D4D_FONT ix);

Input parameters:

- ix—Font index

Output parameters—Font height

Description—Function returns the font height computed from the basic font height and scale in the Y axis of the font defined by the input parameter ix.

D4D_FONT_SIZE D4D_GetFontWidth(D4D_FONT ix);

Input parameters:

- ix—Font index

Output parameters—Font width

Description—Function returns font width computed from the basic font width and scale in the X axis of the font defined by the input parameter ix.

D4D_FONT_SIZE D4D_GetCharWidth(D4D_FONT ix, char ch);

Input parameters:

- ix—Font index
- ch—char

Output parameters—Font width

Description—Function returns char width computed from the current char width and scale in the X axis of the font defined by the input parameter ix.

D4D_FONT_IX D4D_GetCharIndex(D4D_FONT_TYPE* pFontType, char ch);

Input parameters:

- pFontType—Pointer on the D4D_FONT_TYPE structure
- ch—char

Output parameters—Real index to font table

Description—Function returns the index of the char data in the font table for the managed char. The index in the font table is usually different then numeric representation of the char in the ASCII table.

D4D_FONT_DATA* D4D_GetCharData(D4D_FONT_TYPE* pFontType, char ch);

Input parameters:

- pFontType—pointer on the D4D_FONT_TYPE structure
- ch—char

Output parameters—Pointer to the char bitmap data of the used font

Description—Function returns the pointer to the char bitmap data of the used font.

D4D_COOR D4D_GetNextTab(D4D_TAB* pTab, D4D_COOR pos);

Input parameters:

- pTab—Pointer on the D4D_TAB table

- pos—Current position in pixels

Output parameters—Next tabulator position

Description—Function returns the next tabulator position related to current position. Get the tabulator positions from the tabulator table.

3.2.7 D4D Screen

3.2.7.1 Introduction

The D4D_SCREEN is the basic organizational structure of the eGUI/D4D. The whole project structure is divided into individual screens and its objects, therefore screens are used to keep organization of the whole project by the appearance as well as the program structure.

It is recommended to create your own source code file for each individual screen, for example screen_example.c as is shown in Section 2.3.3.1, “User Application eGUI/D4D Dependent Files.”

To simplify the program and user GUI, the driver organization can manage screen histories. It is possible to go through the screens by system forward and back (activate and escape screen).

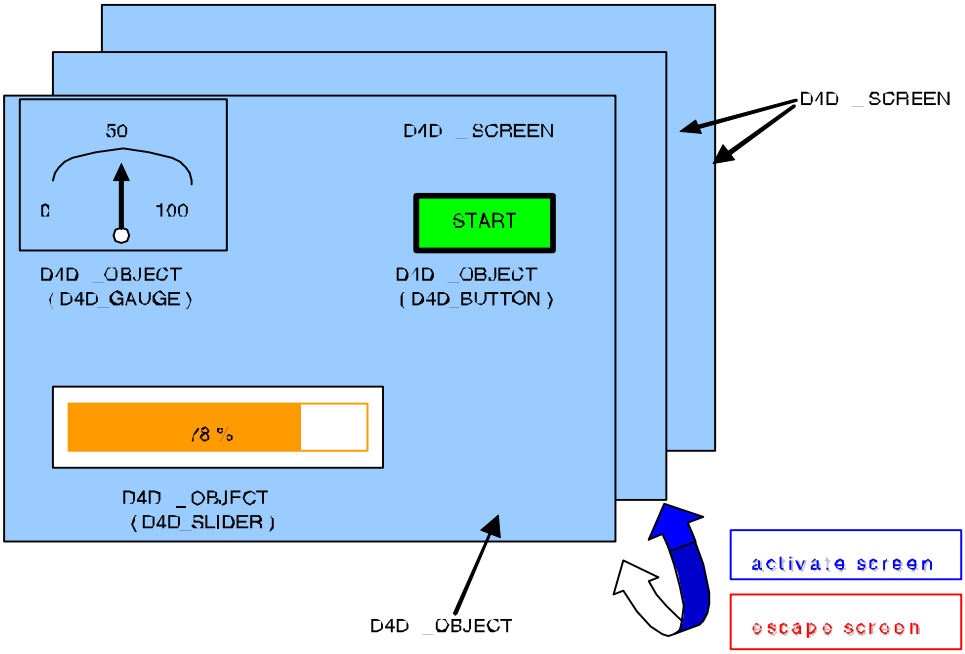


Figure 3-1. D4D structure—objects and screens

The D4D_SCREEN object can be used as a window, have a smaller size, be placed anywhere, and also have a header (title bar) with icon, title, and exit button. Such a screen looks like a window and has many uses (help menu, warning, error, ask, messages, and so on).

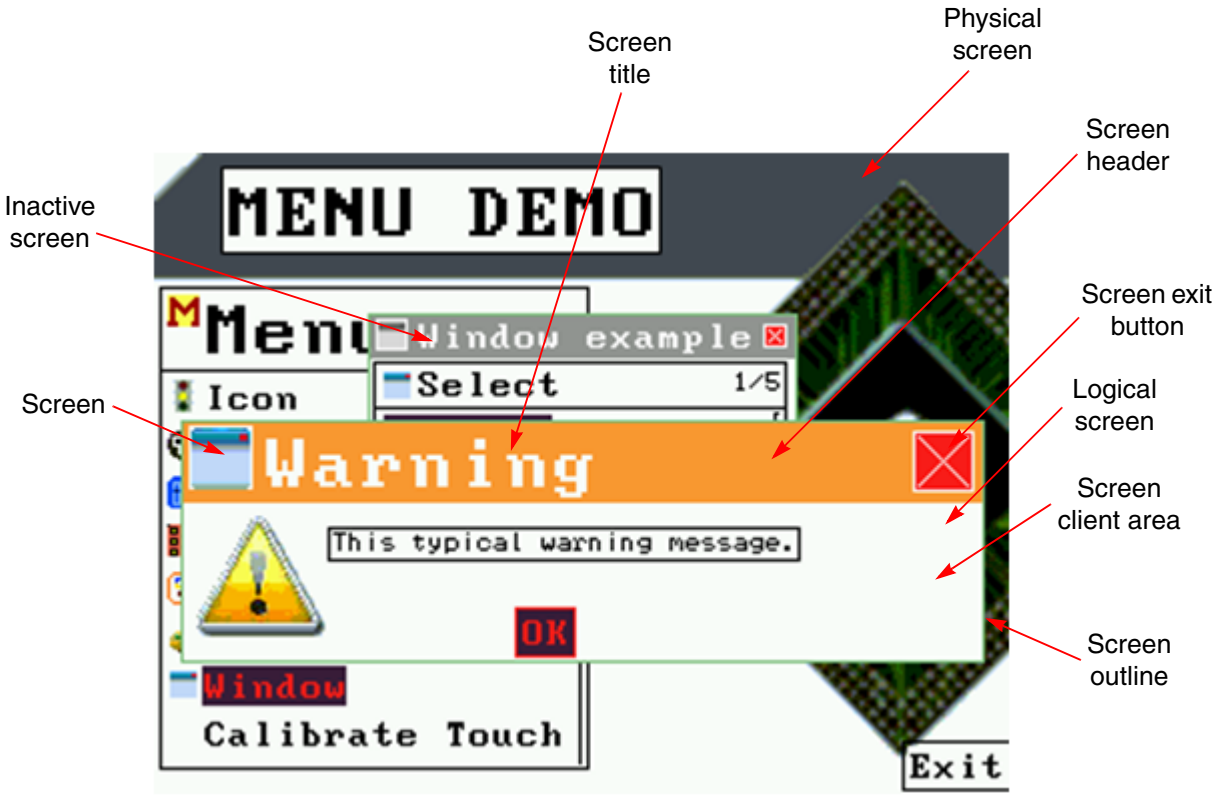


Figure 3-2. The screen with non-default settings

3.2.7.2 D4D_SCREEN Instantiation Macros

The screen definition macro is created from three individual parts that allow to specify the complete screen with objects:

- Begin part—This part specifies all necessary parameters about the screen itself
- Add objects part—This part allows to use multiple D4D_DECLARE_SCREEN_OBJECT macros to add all screen objects.
- End part—This part is used to only close the screen objects array definition.

D4D_DECLARE_SCREEN_BEGIN(name, funcPrefix, x, y, cx, cy, text, fontId, icon, flags, pScheme)

Input parameters:

- name—Name of the screen. Used for working with the screen that follows.
- funcPrefix—Screen mandatory function prefix
- x—Position of the top left corner in axis x
- y—Position of the top left corner in axis y
- cx—Size of the screen in axis x

- cy—Size of the screen in axis y
- text—Text of the screen title
- fontId—Font index of the title text
- icon—Pointer of the icon header bitmap
- flags—Bit mask that sets the behavior and visual aspect of the screen (flags are described in Section 3.2.7.4, “eGUI/D4D Init General Screen Flags”)
- pScheme—Pointer to a color scheme, if NULL, the default color scheme will be used.

Description—This is a full definition macro that allows setting all parameters of the screen individually and defines all the elements of the screen.

D4D_DECLARE_STD_SCREEN_BEGIN(name, funcPrefix)

This macro can be used instead of the previous full screen definition macro in case that the defined screen has standard parameters. The first two input parameters have the same sense as in full begin declaration and the rest are set to these values:

- x = 0, position of the top left corner of the screen in axis X
- y = 0, position of the top left corner of the screen in axis Y
- cx = Physical screen size in axis X
- cy = Physical screen size in axis Y
- text = NULL, screen does not have a title
- fontId = 0, font is not necessary because the title is not used
- icon = NULL, screen does not have an icon
- flags = 0, screen does not have any additional items
- pScheme = NULL, the screen uses the default color scheme

D4D_DECLARE_SCREEN_OBJECT(name)

Input parameters—Name of an added object

Description—This macro can be used multiple times to create a list of all objects that the screen contains. The order of an added object determines the order of focusing the individual objects on the screen when the program runs.

D4D_DECLARE_SCREEN_END()

Description—This macro closes only the screen objects array definition.

D4D_EXTERN_SCREEN(name);

Input parameters—Name of the screen

Description—This is a helper macro that extends the screen name to other files in the project.

3.2.7.2.1 Example of D4D_SCREEN Instantiation Macros Use

Here is an example of the definition of a standard screen with background picture, ten buttons, and two labels.

This example shows how to use all three macros for a screen definition.

```
D4D_DECLARE_STD_SCREEN_BEGIN(screenMainMenu, ScreenMainMenu_)
    D4D_DECLARE_SCREEN_OBJECT(bmp_Background)

    D4D_DECLARE_SCREEN_OBJECT(btn_MM_Gauge)
    D4D_DECLARE_SCREEN_OBJECT(btn_MM_Slider)
    D4D_DECLARE_SCREEN_OBJECT(btn_MM_Mixed)
    D4D_DECLARE_SCREEN_OBJECT(btn_MM_Icon)
    D4D_DECLARE_SCREEN_OBJECT(btn_MM_AnimatedIcon)
    D4D_DECLARE_SCREEN_OBJECT(btn_MM_Menu)
    D4D_DECLARE_SCREEN_OBJECT(btn_MM_KeyPad)
    D4D_DECLARE_SCREEN_OBJECT(btn_MM_Touch)
    D4D_DECLARE_SCREEN_OBJECT(btn_MM_Paint)
    D4D_DECLARE_SCREEN_OBJECT(btn_MM_Window)

    D4D_DECLARE_SCREEN_OBJECT(label_MM_Title)
    D4D_DECLARE_SCREEN_OBJECT(label_MM_Help)
D4D_DECLARE_SCREEN_END()
```

3.2.7.3 D4D_SCREEN Functions

The screen has a few of its own functions that are used to control the screen and its properties.

D4D_SCREEN* D4D_GetActiveScreen(void);

Input parameters—NA

Output parameters—Pointer to an active screen

Description—Function returns a pointer to the current active screen.

void D4D_ActivateScreen(D4D_SCREEN* pNewScreen, Byte bReplaceCurrent);

Input parameters:

- pNewScreen—Pointer to the screen that must be activated
- bReplaceCurrent—A bit that specifies if the new screen replaces the old one in the screen history buffer

Output parameters—NA

Description—This function activates a new screen that is specified in the first parameter. If the second parameter is D4D_TRUE, then the new screen replaces the current screen in the screen history buffer.

void D4D_EscapeScreen(void);

Input parameters—NA

Output parameters—NA

Description—The function escapes from the current active screen and activates the previous screen from the history buffer. If the history buffer is empty, then the function does nothing.

void D4D_InvalidateScreen(D4D_SCREEN* pScreen, Byte bComplete);

Input parameters:

- pScreen—Pointer to the screen that has to be invalidated
- bComplete—This bit specifies if the screen is or is not completely invalidated

Output parameters—NA

Description—This function serves for redrawing the screen. It can be used to clear any user application drawing and return the screen to its original appearance.

void D4D_EnableScrTouchScreen(const D4D_SCREEN* pScr, Byte bEnable);

Input parameters:

- pScreen—Pointer to the screen where settings are to be changed.
- bEnable—This bit specifies if the touch screen events are generated with a touch on the screen.

Output parameters—NA

Description—This function enables the touch screen events when these events are raised anywhere on the current screen client area.

D4D_SIZE D4D_GetClientScreenSize(D4D_SCREEN* pScreen);

Input parameters: pScreen—Pointer to the screen that will be used.

Output parameters—Client screen size

Description—This function returns the client screen size for the managed screen.

3.2.7.4 eGUI/D4D Init General Screen Flags

Each screen in the D4D driver has init flags that indicate screen behavior. The init flags are divided into two groups, first is the appearance group stored in the ROM. The second group of flags are initialization flags for screen behavior.

A list of appearance flags:

- D4D_SCR_F_OUTLINE—Enables screen outline
- D4D_SCR_F_TITLEBAR—Enables screen title bar
- D4D_SCR_F_EXIT—Enables screen title bar exit button
- D4D_SCR_F_BCKG—Enables screen background

A list of behavior flags:

- D4D_SCR_F_TOUCHENABLE—Enables touch screen events for the screen

3.2.7.5 D4D_SCREEN Predefined Standard Object Constants

The driver provides a few predefined colors and the coordination of constants that indicate the visual aspect of the screen. The majority of these constants are used for optional options of the screen. All of these constants can be modified in the user configuration file.

The color definition:

The screen outline rectangle color:

- D4D_COLOR_SCR_OUTLINE (default value—D4D_COLOR_LIGHT_GREY)

The screen title bar color:

- D4D_COLOR_SCR_TITLEBAR (default value: D4D_COLOR_ORANGE)

The screen title bar text color:

- D4D_COLOR_SCR_TITLETEXT (default value: D4D_COLOR_WHITE)

The screen client area background color:

- D4D_COLOR_SCR_DESKTOP (default value: D4D_COLOR_WHITE)

The screen header exit button forecolor:

- D4D_COLOR_SCR_EXIT_BTN_FORE (default value: D4D_COLOR_WHITE)

The screen header exit button background color:

- D4D_COLOR_SCR_EXIT_BTN_BCKG (default value: D4D_COLOR_BRIGHT_RED)

The coordination definition:

The title offset definition in axis X. It is already used for the title icon offset:

- D4D_SCR_TITLE_OFF_X (default value: 3)

The title offset definition in axis Y. It is already used for the title icon offset:

- D4D_SCR_TITLE_OFF_Y (default value: 1)

The header exit button minimal size:

- D4D_SCR_TITLE_EXITBTN_MIN_SIZE (default value: 6)

The header exit button cross size reduction opposite to the total size:

- D4D_SCR_EXITBTN_CROSS_SIZE (default value: 2)

The header exit button minimal size:

- D4D_SCR_TITLE_EXITBTN_OFFSET (default value: 6)

The screen behavior and visual aspect flags:

- D4D_SCR_F_DEFAULT—This is a help macro that is used for default configuration (value: 0).

3.2.8 eGUI/D4D Object

The D4D_OBJECT is the main base item that creates the complete contents of the user screens.

3.2.8.1 D4D_OBJECT Instantiation Macros

D4D_EXTERN_OBJECT(name)

Input parameters—Name of an object

Description—This is a helping macro that extends the object name to other files in the project.

3.2.8.2 eGUI/D4D Init General Object Flags

Each object in the eGUI/D4D driver has init flags that indicate the behavior of initialized objects. The init flags are divided into two groups. The first group is the system group, all the objects have the same flags. Second group of the flags are object dependent.

The list of system init flags:

- D4D_OBJECT_F_VISIBLE—Object after initialization is visible on screen
- D4D_OBJECT_F_ENABLED—Object after initialization is enabled
- D4D_OBJECT_F_TABSTOP—Object can be focused
- D4D_OBJECT_F_TOUCHENABLE—Object has enabled touch screen capability
- D4D_OBJECT_F_FASTTOUCH—Object has enabled fast touch screen capability. This option supports only a few objects (button, check box, and label).
- D4D_OBJECT_F_TRANSP_TEXT—The main text of object is drawn as transparent (without a background color)
- D4D_OBJECT_F_FOCUSRECT—Object has outlined rectangle

3.2.8.3 D4D_OBJECT Functions

All objects have a few common functions that allow to set and control the behavior of individual objects.

void D4D_EnableTabStop (D4D_OBJECT* pObj, Byte bEnable);

Input parameters:

- pObj—Pointer to an object
- bEnable—New status of the tab stop property

Output parameters—NA

Description—This function sets the tab stop property. When the tab stop is enabled, the object can be focused. The initialization state is set by the parameter flags in the object declaration. This function is targeted at using for run-time.

void D4D_EnableObject(D4D_OBJECT_PTR pObj, Byte bEnable);

Input parameters:

- pObj—Pointer to an object
- bEnable—New status of the enabled property

Output parameters—NA

Description—This function sets the enabled property. When the object is enabled, it can accept all user inputs and use normal colors. In the disable state the object is redrawn by grayscale colors and all user inputs are ignored. The initialization state is set by parameter flags in the object declaration. This function is targeted for using at run-time.

void D4D_EnableTouchScreen(D4D_OBJECT_PTR pObj, Byte bEnable, Byte bFastTouch);

Input parameters:

- pObj—Pointer to an object.
- bEnable—New status of the touch screen enable property.
- bFastTouch—New status of the fast touch enable property.

Output parameters—NA

Description—This function sets the touch screen enable and fast touch properties for a given object. When the touch screen is enabled, the object can be touched to receive touch screen events. The second parameter specifies if the object can do a fast touch screen action. For example, the button object changes focus and does a click event by one touch with the touch screen capability. The initialization state is set by parameter flags in the object declaration. This function is targeted for using in run-time.

void D4D_ShowObject(D4D_OBJECT* pObject, Byte bShow);

Input parameters:

- pObj—Pointer to an object.
- bShow—New status of the show property.

Output parameters—NA

Description—This function sets the show property. When the show property is set, the object is visible. The initialization value is set by parameter flags in the object declaration. This function is targeted to be used for run-time.

D4D_OBJECT* D4D_GetFocusedObject(D4D_SCREEN* pScreen);

Input parameters—Pointer to a screen

Output parameters—Pointer to the current focused object

Description—The function returns the pointer to the focused object on the screen which is managed by the input parameter.

void D4D_FocusSet(D4D_SCREEN* pScreen, D4D_OBJECT_PTR pObject);

Input parameters:

- Pointer to a screen
- Pointer to the object that must be focused

Output parameters—NA

Description—The function sets the focus to a new object that is managed by parameter pObject on the screen which is managed by the parameter pScreen. The screen and object must exist and the object has to have a tab stop property enabled.

void D4D_InvalidateObject(D4D_OBJECT* pObject, Byte bComplete);

Input parameters:

- pObject—Pointer to an object that must be invalidated
- bComplete—Extent of invalidation

Output parameters—NA

Description—The function invalidates the object, the object is redrawn. The bComplete parameter specifies the range of invalidation, if it sets and the selected object is completely redrawn.

void* D4D_GetUserPointer(D4D_OBJECT *pThis);

Input parameters:

- pObject—Pointer to an object

Output parameters—User general pointer

Description—This function returns user pointer of the selected object. Each object contains user pointer (void*) that may be defined by a non-standard object declaration. This pointer is designed for user general use.

void D4D_CaptureKeys(D4D_OBJECT_PTR pObj);

Input parameters:

- pObject—Pointer to an object

Output parameters—NA

Description—This function sets the object to the capture keys state. In this state the object obtains all the keys inputs including system navigation keys (escape, up, and down). In this state the object is using capture colors from a color scheme. To switch off from this state the active screen has to be changed or this function has to be called with the input parameter set to NULL.

D4D_OBJECT* D4D_GetCapturedObject(void);

Input parameters—NA

Output parameters—Pointer to a captured object

Description—This function returns the pointer to the current object and sets it as a capture keys object. If neither object is set as capture keys, the function returns NULL.

3.2.9 eGUI/D4D Base of Graphical Objects

The eGUI/D4D contains a few prepared graphical objects that allow to create various combinations of screens that allow control of many applications, show status, and many other user applications.

NOTE

All objects contain instantiation macros that create gripped objects mostly in ROM memory and only runtime data in the RAM (see Figure 2-4). All of these macros have others with same name and postfix *_INRAM*. The macros with *_INRAM* postfix creates complete graphic objects in RAM memory and thus all parameters can change runtime, but the screen must be redrawn manually after each change.

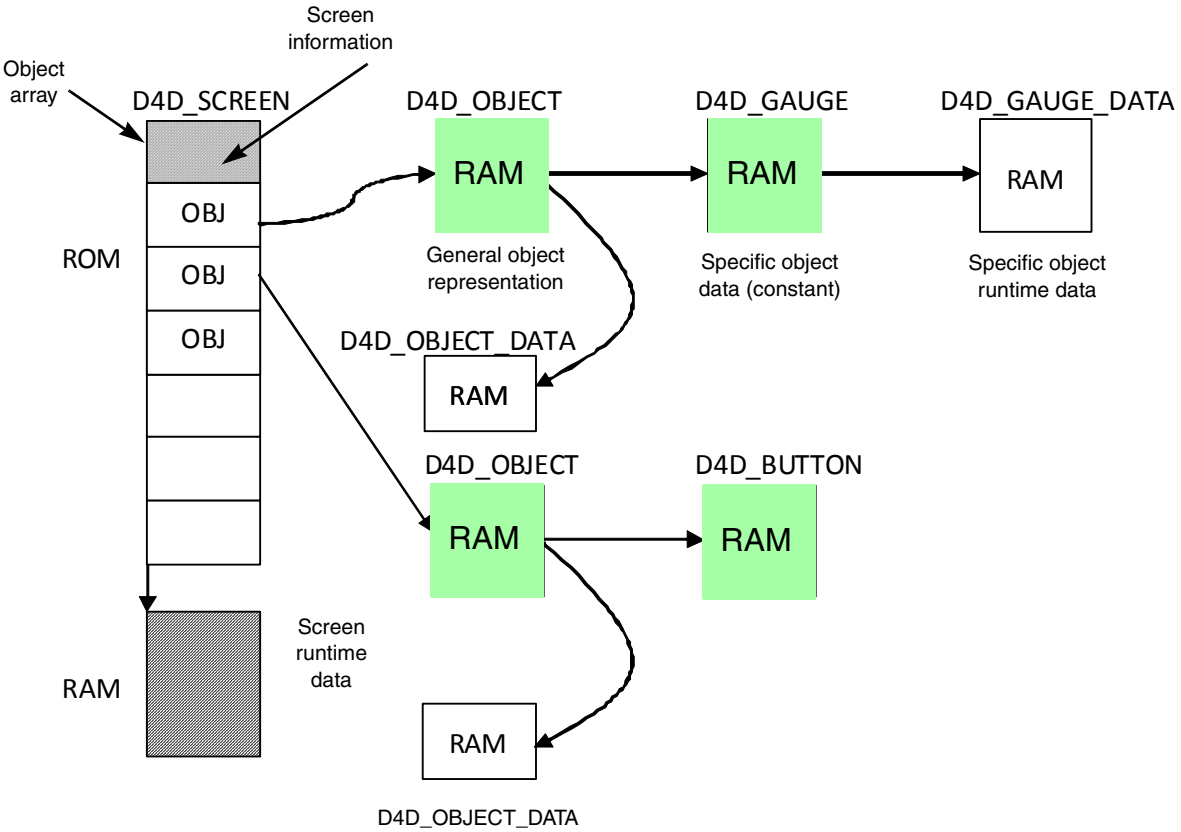


Figure 3-3. Memory map for screens and objects created in RAM memory by _INRAM postfix instantion macros

3.2.9.1 D4D_BUTTON

The D4D_BUTTON object is intended to be used as a standard button providing many basic features:

- Variable position, size, and text position
- Title text that can be run-time changed
- Underline, strike through and transparent text option
- Disable/enable capability

- Two background bitmaps for focused and normal state
- Auto size capability for size and title text offset
- Touch screen support to focus and click action with fast touch capability (both actions on one touch)
- Variable colors of text and background in normal and focus state
- OnClick callback function to simplify the user application
- pOnUsrMsg —Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

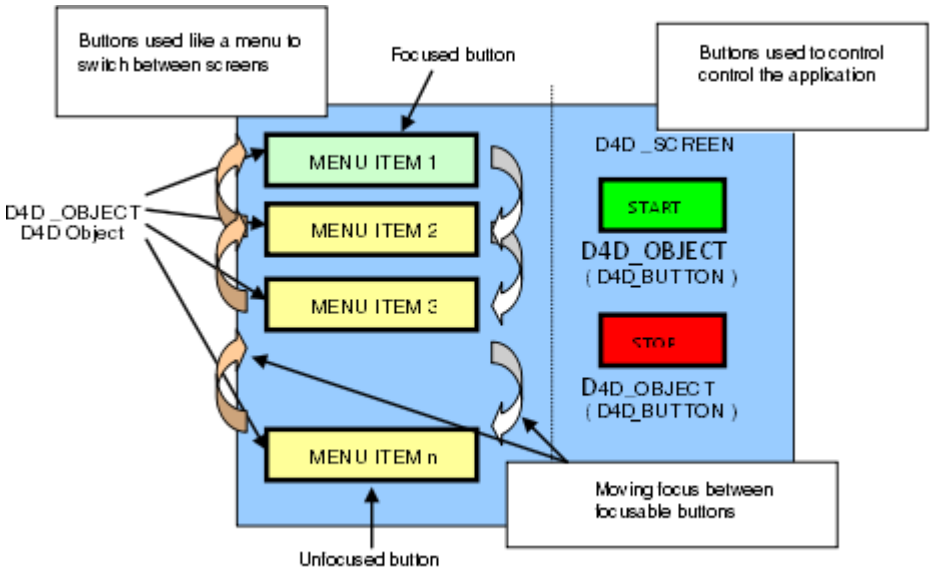


Figure 3-4. Button objects on the screen

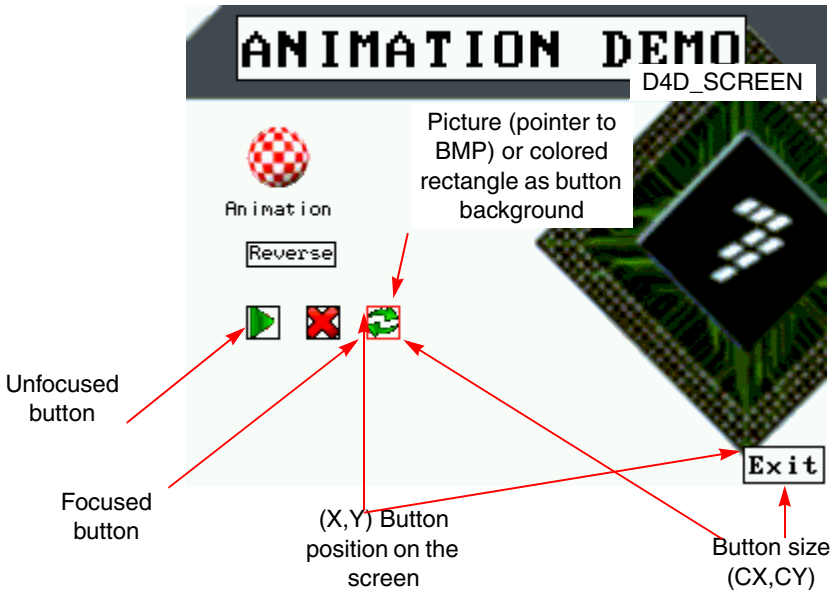


Figure 3-5. Button location and description

3.2.9.1.1 D4D_BUTTON Instantiation Macros

The button object has an instantiation macro that specifies all its parameters. The driver contains one full declaration macro that allows to set all parameters of the button object and a few modifications of this macro that simplifies the full declaration by default object values or by auto size capability.

NOTE

All button declaration macros have a second version with the letter **R**—**RBUTTON** instead of **BUTTON** is used to create button objects with rounded corners. In these new macros there is an add new parameter "radius" that specifies the radius of the rounded corners.

D4D_DECLARE_BUTTON(name, text, x, y, cx, cy, flags, pbmpN, pbmpF, pScheme, fontId, pUser, onclick, pOnUsrMsg)

Input parameters:

- name—Name of a button object
- text—Title text of a button
- x—Position of a button on the client area of the screen in axis X
- y— Position of a button on the client area of the screen in axis Y
- cx—Size of a button in axis X, if this value is 0, the driver computes the size of the button automatically
- cy—Size of a button in axis Y, if this value is 0, the driver computes the size of the button automatically
- flags—Bitmask that specifies initial system object flags and button object flags

- pbmpN—Pointer to a bitmap that is shown in a normal state of the button
- pbmpF—Pointer to a bitmap that is shown in a focused state of the button
- pScheme—Pointer to a color scheme, if it is NULL, the default scheme is used
- fontId—Identification number of the used title text
- pUser—User defined pointer to void, can be used by the user application
- onclick—Pointer to an on-click user callback function (in format: “void (*OnClicked)(D4D_OBJECT* pThis)”)
- pOnUsrMsg—Pointer to an on user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal case the return value must be D4D_MSG_NOSKIP

Description—This is a full definition macro that allows setting of all the parameters of the button object individually and allows to modify all button options.

D4D_DECLARE_STD_BUTTON(name, text, x, y, cx, cy, bmpN, bmpF, fontId, onclick)

Input parameters—All parameters in the declaration have the same sense as in a full declaration macro. The rest of the parameters have default values:

- flags—Has the value of macro D4D_BTN_F_DEFAULT.
- pScheme—Has the NULL value, the default color scheme will be used.
- pUser—Has a NULL value.
- pOnUsrMsg—Has a NULL value

Description—This is the default button definition. The main advantage is less parameters of this macro against the full version.

D4D_DECLARE_TXT_BUTTON(name, text, x, y, cx, cy, fontId, onclick)

Input parameters—Input parameters are the same as in a standard button definition, only the bitmaps parameters (pbmpN, pbmpF) are missing and set to NULL as a default state.

Description—This is a simplified definition of a simple button with text only, without any bitmaps.

D4D_DECLARE_TXT_BUTTON _AUTOSIZE(name, text, x, y, fontId, onclick)

Input parameters—Input parameters are the same as in the text button definition, only size parameters are replaced by zero, this invokes auto size capability for these parameters.

Description—This is a definition that has as few as possible parameters to configure a button object. The size values are generated run-time from the size of the used font and title text.

D4D_DECLARE_STD_BUTTON_AUTOSIZE(name, text, x, y, bmpN, bmpF, fontId, onclick)

Input parameters—Input parameters are the same as in a standard button definition, only size parameters are replaced by zero, this invokes auto size capability for this parameters.

Description—This is a definition that has as few as possible parameters to configure a standard button object. The size values are generated run-time from the size of the used font and title text.

3.2.9.1.2 D4D_BUTTON Predefined Standard Constants

The button object contains a few predefined constants that are used in a standard button declaration and helps to indicate the resulting sizes when the auto size is enabled. All of these constants can be modified in the user configuration file.

The coordination definition:

The border offset definition in axis X and Y

- D4D_BTN_BORDER_OFFSET (default value: 3)

The screen behavior and visual aspect flags:

- D4D_BTN_F_DEFAULT—This is a help macro that is used for default configuration (value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLE | D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FOCUSRECT).
- D4D_BTN_TXT_PRTY_DEFAULT—This is a text property macro and is used for default configuration of the button text (value: D4D_TXT_PRTY_ALIGN_H_CENTER_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).
- D4D_BTN_FNT_PRTY_DEFAULT—This is a font property macro and is used for default configuration of button font (value: 0).

3.2.9.1.3 D4D_BUTTON Functions

void D4D_BtnSetText(D4D_OBJECT_PTR pObj, char* pText);

Input parameters:

- pObj—Pointer to an object.
- pText—Pointer to a new string.

Output parameters—NA

Description—Function changes the title text of a button. To have success running this function, the original declared text in instantiation macro has to be placed in RAM.

NOTE

This function is not preferred because version 1.0 and the direct replacement is general function D4D_SetText with the same parameters and behavior.

3.2.9.1.4 Example of Use D4D_BUTTON

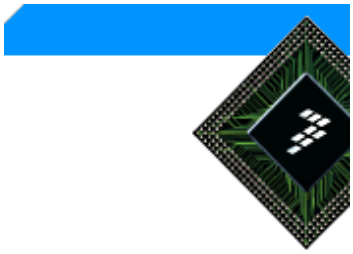
```
static void OnClicked_Btn1(D4D_OBJECT* pThis);

D4D_DECLARE_BMP(scr1_bmpButton1Off, btn1_ina, NULL)
```

```
D4D_DECLARE_BMP(scr1_bmpButton1On, btn1_act, NULL)
```

```
D4D_DECLARE_STD_BUTTON(scr1_btn1, “BUTTON”, 5, 29, 129, 22, 20, 4, &scr1_bmpButton1Off,
&scr1_bmpButton1On, fontId, OnClicked_Btn1)
```

```
D4D_DECLARE_SCREEN_BEGIN(screenBtn, ScreenButton_)
D4D_DECLARE_SCREEN_OBJECT(scr1_back)
D4D_DECLARE_SCREEN_OBJECT(scr1_btn1)
```



```
D4D_DECLARE_SCREEN_END()
static void OnClicked_Btn1(D4D_OBJECT* pThis)
{
    // Does something when the button is clicked
}
```

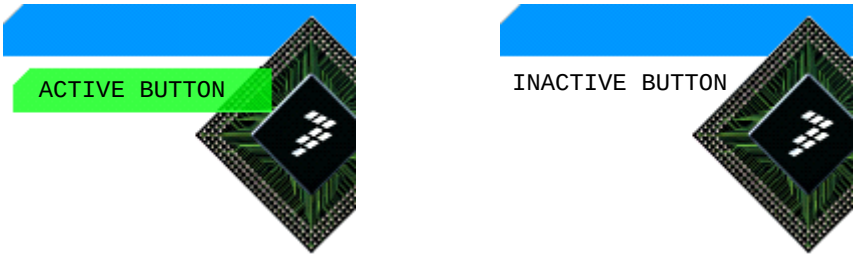


Figure 3-6. Focused and unfocused button

3.2.9.2 D4D_CHECKBOX

The D4D_CHECKBOX object is prepared to be used as a standard checkbox known from a PC. A user-managed radio button provides many basic features:

- Variable position and size
- User bitmaps for check icon (checked and unchecked state)
- Disable and enable capability
- Text that run-time can be changed
- Underline, strike through and transparent text option
- Auto size capability for size
- Touch screen support to focus and change actions with fast touch capability (both actions in one touch)
- Variable colors of text and background in normal and focus state
- OnChange callback function to simplify user application
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

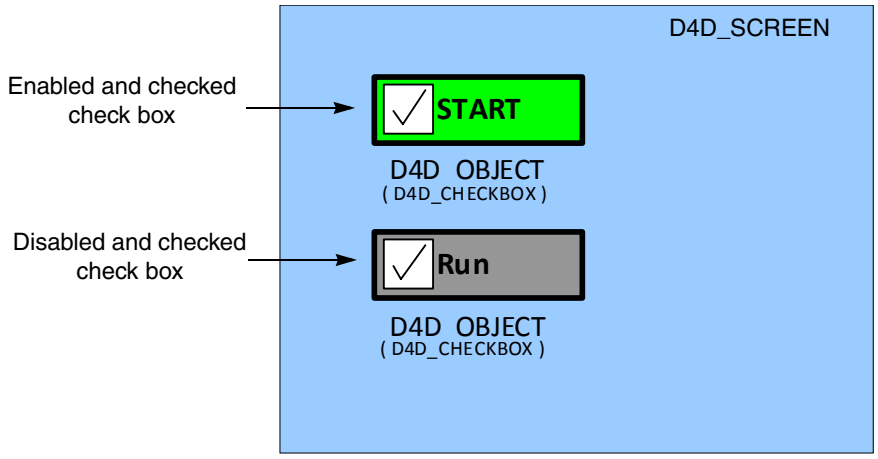


Figure 3-7. CHECK Box objects on the screen

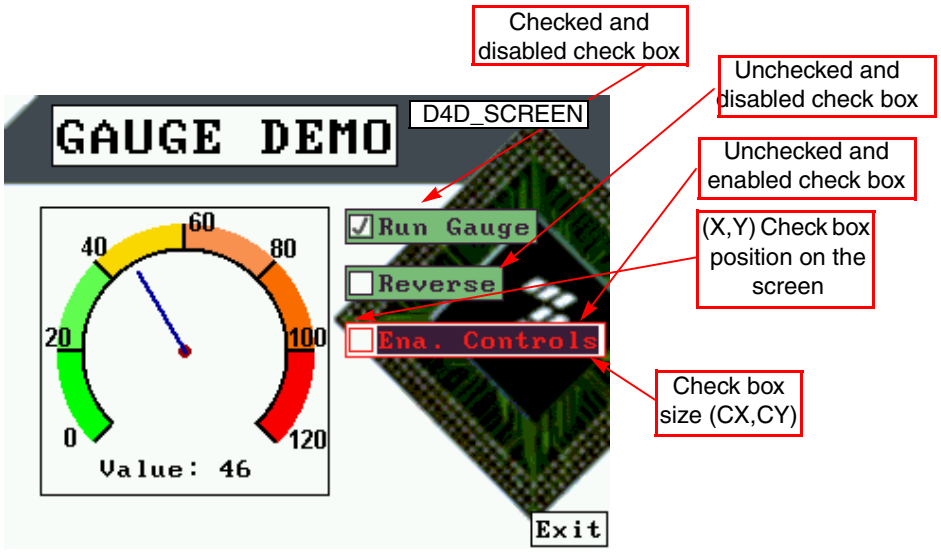


Figure 3-8. CHECK BOX location and description

3.2.9.2.1 D4D_CHECKBOX Instantiation Macros

The checkbox object has an instantiation macro that specifies all its variable parameters. The driver contains one full declaration macro that allows to set all parameters of the checkbox object and a few modifications of this macro that simplifies the full declaration by default object values or by auto size capability.

D4D_DECLARE_CHECKBOX(name, text, x, y, cx, cy, flags, pbmpChecked, pbmpUnChecked, pScheme, fontId, pUser, onchange, pOnUsrMsg)

Input parameters:

- name—Name of the checkbox object
- text—Text of the checkbox
- x—Position of the checkbox in the client area of the screen in axis X
- y—Position of the checkbox in the client area of the screen in axis Y
- cx—Size of the checkbox in axis X, if this value is zero, the driver computes the size of the button itself
- cy— Size of the checkbox in axis Y, if this value is zero, the driver computes the size of button itself
- flags—Bitmask that specifies the initial system object flags and the checkbox object flags
- pbmpChecked—Pointer to a bitmap that is shown in the checked state of the checkbox
- pbmpUnChecked—Pointer to a bitmap that is shown in the unchecked state of the checkbox
- pScheme—Pointer to a color scheme, if it is NULL, the default scheme is used
- fontId—Identification number of the used font of the text
- pUser—User pointer on void that can be used by user application any way the user wants
- onchange—Pointer to the on-click user callback function (in format: “void (*OnChange)(D4D_OBJECT* pThis)”)
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP

Description—This is a full definition macro that allows setting all the parameters of a checkbox object individually, it allows modifying all the checkbox options.

D4D_DECLARE_STD_CHECKBOX(name, text, x, y, cx, cy, pbmpChecked, pbmpUnChecked, fontId, onchange)

Input parameters—All parameters in the declaration have the same purpose as in the full declaration macro, the rest of the parameters have default values:

- flags—Has the value of macro D4D_CHECKBOX_F_DEFAULT.
- pScheme—Has the NULL value, the default color scheme will be used.
- pUser—Has the NULL value.
- pOnUsrMsg—Has the NULL value.

Description—This is the default checkbox definition. The main advantage is less parameters of this macro against one full version.

D4D_DECLARE_STD_CHECKBOX_AUTOSIZE(name, text, x, y, pbmpChecked, pbmpUnChecked, fontId, onchange)

Input parameters—Input parameters are the same as in a standard checkbox definition, only size parameters are replaced by zero that invoke auto size capability for these parameters.

Description—This is the definition that has as few parameters as possible to configure a standard checkbox object. The size values are generated run-time from the size of the used bitmaps, font, and title text.

3.2.9.2.2 D4D_CHECKBOX Predefined Standard Constants

The checkbox object contains a few predefined constants that are used in a standard checkbox declaration. It helps to indicate the result sizes when the auto size is enabled. All of these constants can be modified in the user configuration file.

The color definition:

The checkbox icon background color:

- D4D_COLOR_CHECKBOX_ICON_BCKGN (default value: D4D_COLOR_WHITE)

The coordination definition:

The border offset definition in axes X and Y:

- D4D_CHECKBOX_BORDER_OFFSET (default value: 3)

The text offset definition in axes X and Y:

- D4D_CHECKBOX_TEXT_OFFSET (default value: 4)

The screen behavior and visual aspect flags:

The checkbox icon border enables flag:

- D4D_CHECKBOX_F_ICON_RECTANGLE—This bit enables the checkbox icon outline.
- D4D_CHECKBOX_F_DEFAULT—This is a help macro used for default configuration (value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLE | D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FOCUSRECT | D4D_CHECKBOX_F_ICON_RECTANGLE).
- D4D_CHECKBOX_TXT_PRTY_DEFAULT—This is a text property macro used for default configuration of check box text (value: D4D_TXT_PRTY_ALIGN_H_LEFT_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).
- D4D_CHECKBOX_FNT_PRTY_DEFAULT—This is a font property macro used for default configuration of check box font (value: 0).

3.2.9.2.3 D4D_CHECKBOX Functions

void D4D_CheckBoxSetValue(D4D_OBJECT_PTR pThis, Byte value);

Input parameters:

- pThis—Pointer to a checkbox object
- value—New value of the checkbox

Output parameters—NA

Description—The function sets the new value of a checkbox.

Byte D4D_CheckBoxGetValue(D4D_OBJECT_PTR pThis);

Input parameters—pThis—Pointer to a checkbox object

Output parameters—Current set value of a checkbox

Description—Function returns the current state of a checkbox object

void D4D_ D4D_CheckBoxSetText (D4D_OBJECT_PTR pObj, char* pText);

Input parameters:

- pObj—Pointer to an object.
- pText—Pointer to a new string.

Output parameters—NA

Description—Function changes the title text of the checkbox. To have success running this function, the original declared text in the instantiation macro has to be placed in the RAM.

NOTE

This function is not preferred because version 1.0 and the direct replacement is general function D4D_SetText with the same parameters and behavior.

3.2.9.2.4 Example of D4D_CHECKBOX

```
static void OnChange_CheckB1(D4D_OBJECT* pThis);

D4D_DECLARE_BMP(scr1_bmpCheckBChecked, CheckB_chck, NULL)
```

```
D4D_DECLARE_BMP(scr1_bmpCheckBUNChecked, CheckB_unchck, NULL)
```

```
D4D_DECLARE_STD_CHECKBOX_AUTOSIZE(scr1_checkB1, "Ena. Controls", 180, 60, &
scr1_bmpCheckBChecked, & scr1_bmpCheckBUNChecked, FONT_8x14, OnChange_CheckB1)

D4D_DECLARE_SCREEN_BEGIN(screenChckB, ScreenCheckBox_)
D4D_DECLARE_SCREEN_OBJECT(scr1_checkB1)
D4D_DECLARE_SCREEN_END()
```


```
static void OnChange_CheckB1 (D4D_OBJECT* pThis)
{
    // Does something when the check box is changed
}
```

3.2.9.3 D4D_GAUGE

The D4D_GAUGE object is prepared to be used as a visualization object of any numerical value that provides many basic features:

- Variable position, size, hub position, pointer length, and text position

- Title text that can be run-time changed
- Underline, strike through and transparent text option
- Disable/enable capability
- Background bitmap that allows making various gauges
- Touch screen support to focus
- Variable colors of text, hub, pointer, and background
- Different values for angle range setting and input control value range (angle is unsigned variable, value is signed variable)
- Direction of rotation control
- OnValueChange callback function to simplify user application
- pOnUsrMsg —Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP

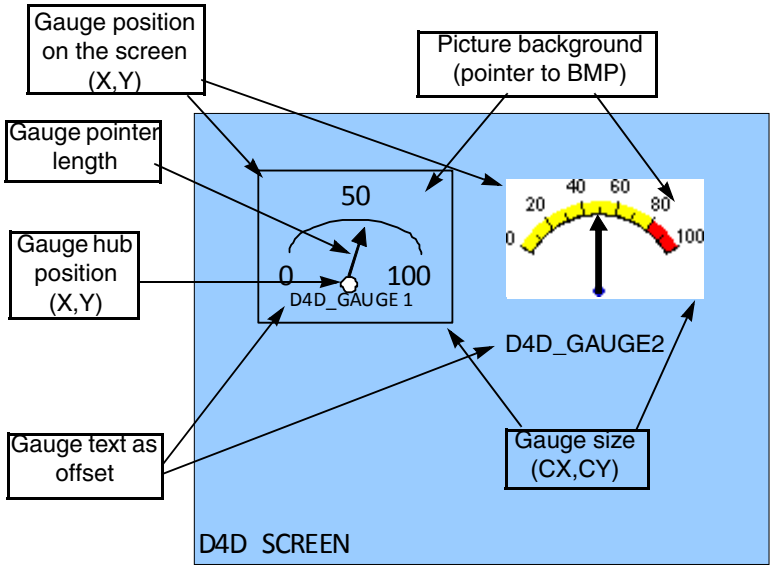


Figure 3-9. GAUGE location and description

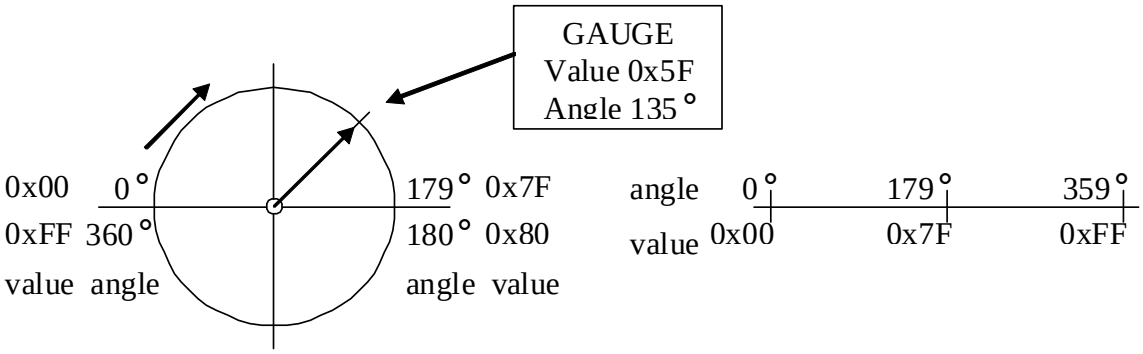


Figure 3-10. GAUGE angle representation by an 8-bit unsigned variable

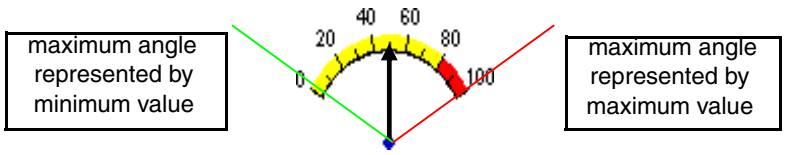


Figure 3-11. GAUGE limits

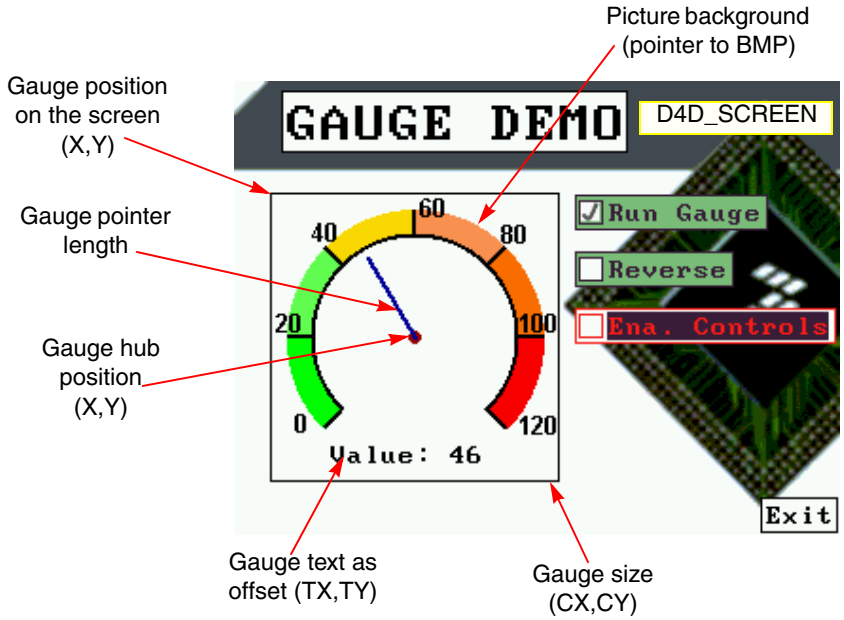


Figure 3-12. GAUGE example

3.2.9.3.1 D4D_GAUGE Instantiation Macros

The gauge object has a simple instantiation macro that specifies all its variable parameters. The driver contains one full declaration macro that allows setting all parameters of gauge object and one modification of this macro that simplifies the full declaration by default object values.

D4D_DECLARE_GAUGE(name, text, x, y, cx, cy, tx, ty, kx, ky, plen, flags, pBmp, pScheme, fontId, pUser, onvalch, pOnUsrMsg)

Input parameters:

- name—The name of a gauge object
- text—Title text of a gauge
- x—Position of a gauge on the client area of the screen in axis X
- y—Position of a gauge in the client area of screen in axis Y
- cx—Size of a gauge in axis X
- cy—Size of a gauge in axis Y
- tx—Offset of the title in axis X
- ty—Offset of the title in axis Y
- kx—Offset of the hub in axis X

- ky—Offset of the hub in axis Y
- plen—Length of the gauge pointer
- flags—Bitmask that specifies the initial system object flags and gauge object flags
- pbmp—Pointer to a bitmap that is shown on the background of gauge
- pScheme—Pointer to a color scheme, if it is NULL, the default scheme is used
- fontId—Identification number of the used font
- pUser—Pointer on void, that may be used by user application any way the user wants
- onvalch—Pointer to the on-value change user callback function (in format: “void (*OnValueChanged)(D4D_OBJECT_PTR pThis)”)
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

Description—This macro creates a complete gauge object with individually setting all parameters of this object.

D4D_DECLARE_STD_GAUGE(name, text, x, y, cx, cy, tx, ty, kx, ky, plen, pBmp, font, onvalch)

Input parameters—All parameters in the declaration have the same purpose as in the full declaration macro, the remaining of the parameters have default values:

- flags—Has value of macro D4D_GAUGE_F_DEFAULT
- pScheme—Has value NULL, the default color scheme will be used
- pUser—Has NULL value
- pOnUsrMsg—Has NULL value

Description—This is the default gauge definition. The main advantage is less parameters of this macro against one full version.

3.2.9.3.2 D4D_GAUGE Types

The D4D_GAUGE object has its own type that it needs to run on.

A list of Gauge object data types:

- D4D_GAUGE_VALUE—Input control value of gauge object. Type is signed char.
- D4D_GAUGE_ANGLE—A variable that specifies the angles of gauge object. It is used to set pointer range limits. Type is unsigned char.
- D4D_GAUGE_DIR—A variable that specifies the direction of gauge. It can have two values D4D_CLOCK_WISE or D4D_ANTI_CLOCK_WISE. Type is unsigned char.
- D4D_GAUGE_LIMITS—A structure type that contains information about limit values of angle and input control value.

```
typedef struct {
    D4D_GAUGE_VALUE valueMin;   // minimal value (corresponds to angleMin)
    D4D_GAUGE_VALUE valueMax;   // maximal value (corresponds to angleMax)
    D4D_GAUGE_ANGLE angleMin;             // line angle from 0x00 to 0xFF
```

```
    D4D_GAUGE_ANGLE angleMax;             // line angle from 0x00 to 0xFF
} D4D_GAUGE_LIMITS;
```

3.2.9.3.3 D4D_GAUGE Predefined Standard Constants

The gauge object contains a few predefined constants that are used in a standard gauge declaration. All of these constants could be modified in the user configuration file.

The color definition:

The gauge hub color:

- D4D_COLOR_GAUGHUB (default value: D4D_COLOR_RED)

The gauge pointer color:

- D4D_COLOR_GAUGPOINTER (default value: D4D_COLOR_BLUE)

The coordination definition:

The default hub radius:

- D4D_GAUGE_HUB_RADIUS (default value: 3)

The screen behavior and visual aspect flags:

- D4D_GAUGE_F_REDRAW_TEXT—This flag specifies that text is always redrawing. This option is useful when the text is placed into the pointer range area.
- D4D_GAUGE_F_HUB—This flag enables gauge hub drawing.
- D4D_GAUGE_F_THICK_POINTER—This flag specifies the thickness of the pointer, when it is set, and the pointer is thick.
- D4D_GAUGE_F_DEFAULT—This is a help macro that is used for default configuration (value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLE | D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FOCUSRECT | D4D_GAUGE_F_REDRAW_TEXT | D4D_GAUGE_F_HUB).
- D4D_GAUGE_TXT_PRTY_DEFAULT—This is a text property macro used for default configuration of gauge text (value: D4D_TXT_PRTY_ALIGN_H_CENTER_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).
- D4D_GAUGE_FNT_PRTY_DEFAULT—This is a font property macro used for default configuration of gauge font (value: 0).

3.2.9.3.4 D4D_GAUGE Functions

void D4D_GaugSetValue(D4D_OBJECT_PTR pThis, D4D_GAUGE_VALUE value);

Input parameters:

- pThis—Pointer to a gauge object
- value—New value of a gauge, it must fit into the range set in D4D_GAUGE_LIMITS.

Output parameters—NA

Description—The function sets the new value of gauge.

D4D_GAUGE_VALUE D4D_GaugGetValue(D4D_OBJECT_PTR pThis);

Input parameters—pThis—Pointer to a gauge object

Output parameters—Current set value of gauge

Description—Function returns the current value of a gauge object

void D4D_GaugSetLimits(D4D_OBJECT_PTR pThis, const D4D_GAUGE_LIMITS* pLimits);

Input parameters:

- pThis—Pointer to a gauge object
- pLimits—Pointer to a new D4D_GAUGE_LIMITS structure that is used as new in the gauge object

Output parameters—NA

Description—The function loads a new limits value to the gauge object and also takes care of adapting the current direction and value of a gauge to fit into the new limits range.

void D4D_GaugGetLimits(D4D_OBJECT_PTR pThis, D4D_GAUGE_LIMITS* pLimits);

Input parameters:

- pThis—Pointer to a gauge object.
- pLimits—Pointer to the D4D_GAUGE_LIMITS structure in a volatile memory used to store current limit value of the gauge object.

Output parameters—NA

Description—The function copy current sets limits of the gauge object to the D4D_GAUGE_LIMITS structure in the volatile memory.

void D4D_GaugSetDir(D4D_OBJECT_PTR pThis, D4D_GAUGE_DIR direction);

Input parameters:

- pThis—Pointer to a gauge object
- direction—A new value of the direction of a gauge object

Output parameters—NA

Description—The function sets the new direction of movement for a gauge pointer.

D4D_GAUGE_DIR D4D_GaugGetDir(D4D_OBJECT_PTR pThis);

Input parameters—pThis—Pointer to a gauge object

Output parameters—Currently uses the direction of a gauge object

Description—The function returns the current direction of movement for the gauge pointer

void D4D_GaugeSetText(D4D_OBJECT_PTR pObj, char* pText);

Input parameters:

- pObj—Pointer to a gauge object.
- pText—Pointer to a new string.

Output parameters—NA

Description—Function changes the title text of the gauge. To have success running this function, the original declared text in the instantiation macro has to be placed in RAM.

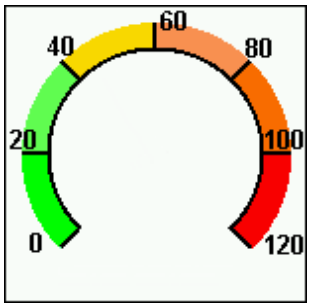
NOTE

This function is not preferred because version 1.0 and direct replacement is the general function D4D_SetText with the same parameters and behavior.

3.2.9.3.5 Example of Use D4D_GAUGE

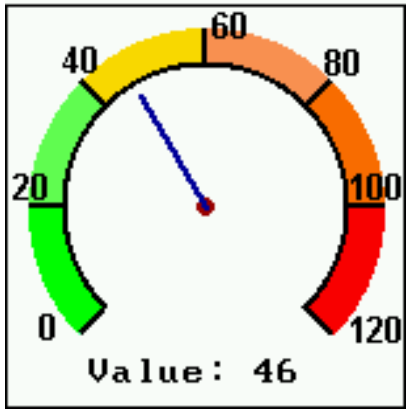
```
static void OnChange_Gauge1(D4D_OBJECT* pThis);

D4D_DECLARE_BMP(scr1_bmpGauge, gaugeBmp, NULL)
```




```
D4D_DECLARE_STD_GAUGE(scr1_Gauge1,"Value: 46", 0, 0, 153, 153, 30, 130, 0, 0, 50, &
scr1_bmpGauge, FONT_8x14, OnChange_Gauge1)

D4D_DECLARE_SCREEN_BEGIN(screengauge, ScreenGauge_)
D4D_DECLARE_SCREEN_OBJECT(scr1_Gauge1)
D4D_DECLARE_SCREEN_END()
```



```
static void OnChange_Gauge1 (D4D_OBJECT* pThis)
{
    // Does something when the gauge value is changed
}
```

3.2.9.4 D4D_PICTURE

The D4D_PICTURE is an object used to show a picture on the display that provides the following features:

- Variable position of a shown bitmap
- Disable/enable capability
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

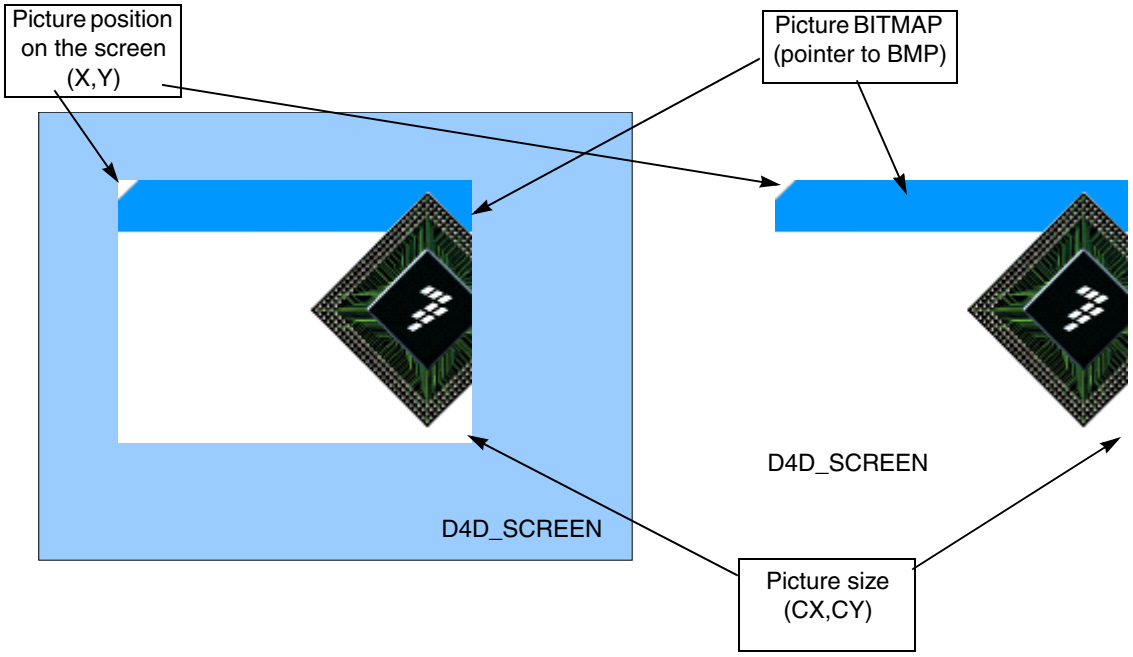


Figure 3-13. PICTURE on the screen and screen represented by PICTURE

3.2.9.4.1 D4D_PICTURE Instantiation Macros

The picture object has a simple instantiation macro that specifies all of its variable parameters. The driver contains one full declaration macro that allows setting all parameters of the picture object and one modification of this macro that simplifies the full declaration by default object values.

#define D4D_DECLARE_PICTURE(name, x, y, pBmp, flags, pUser, pOnUsrMsg)

Input parameters:

- name—name of a picture object
- x—Position of a picture in the client area of the screen in axis X

- y—Position of a picture in the client area of the screen in axis Y
- flags—Bitmask that specifies the initial system object flags
- pUser—Pointer on void, can be used by user application any way the user wants
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

Description—This macro creates a complete picture object by individually setting all parameters of this object.

#define D4D_DECLARE_STD_PICTURE(name, x, y, pBmp)

Input parameters—All parameters in the declaration have the same purpose as in a full declaration macro, the rest of the parameters have default values.

- flags—Has the value of macro D4D_PIC_F_DEFAULT.
- pUser—Has NULL value.
- pOnUsrMsg—Has NULL value.

Description—This is the default picture definition. The main advantage is less parameters of this macro against one full version.

3.2.9.4.2 D4D_PICTURE Predefined Standard Constants

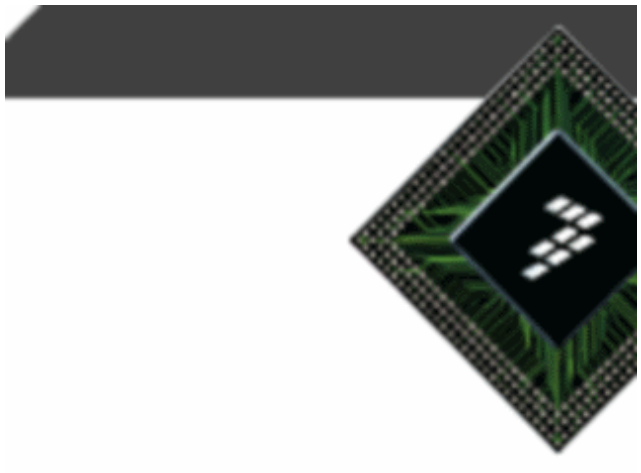
The picture object contains only one predefined constant that is used in a standard picture declaration. This constant can be modified in the user configuration file.

The screen behavior and visual aspect flags:

- D4D_PIC_F_DEFAULT—This is a help macro that is used for default configuration (value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLE).

3.2.9.4.3 Example of Use D4D_PICTURE

```
D4D_DECLARE_BMP(scr1_bmpPicture, pictureBmp, NULL)
```



```
D4D_DECLARE_STD_PICTURE(scr1_Picture1, 0, 0, & scr1_bmpPicture)

D4D_DECLARE_SCREEN_BEGIN(screenpicture, ScreenPicture_)
D4D_DECLARE_SCREEN_OBJECT(scr1_Picture1)
D4D_DECLARE_SCREEN_END()
```

3.2.9.5 D4D_ICON

The D4D_ICON object is prepared to be used as a visualization object to show various states of the application that provide many basic features:

- Variable position, size, and text position
- Title text that can be run-time changed
- Underline, strike through and transparent text option
- Disable/enable capability
- Bitmap array
- Easy bitmap change system
- Touch screen support to focus
- Variable colors of text and background
- OnValueChange callback function to simplify user application
- pOnUsrMsg —Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

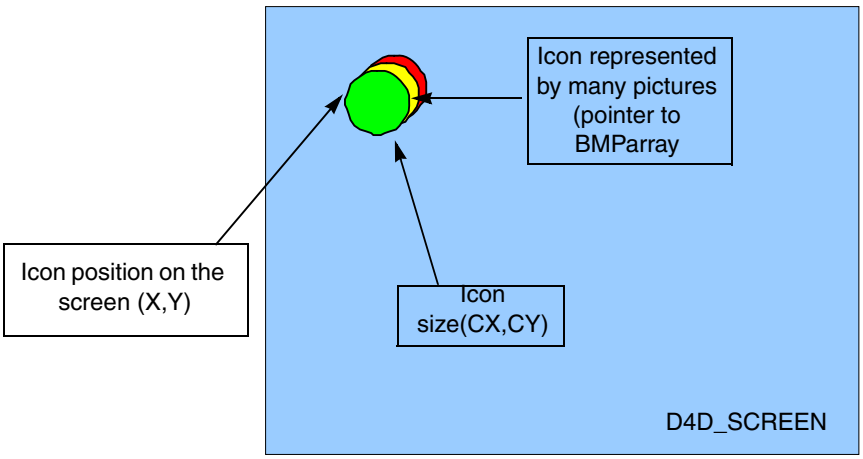


Figure 3-14. ICON location and description

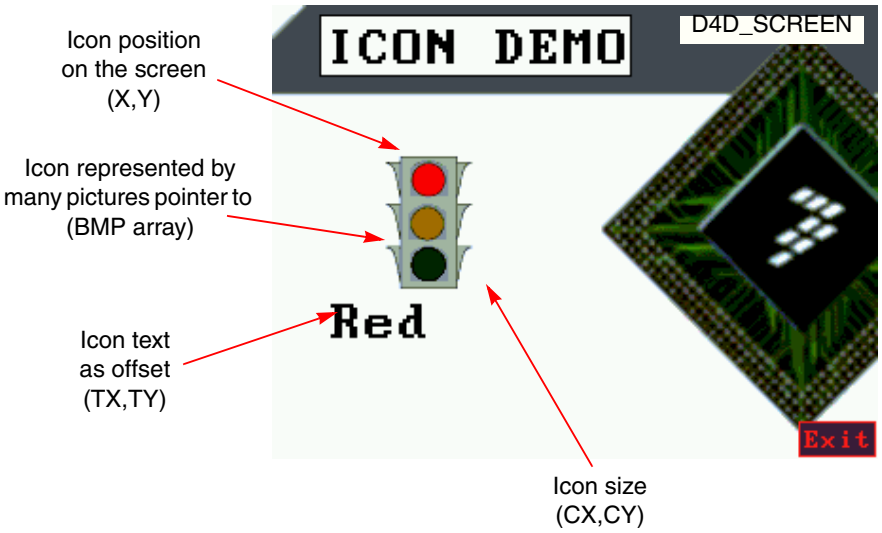


Figure 3-15. ICON location and description example

3.2.9.5.1 D4D_ICON Instantiation Macros

The icon object has an instantiation macro in the same shape as the screen declaration macro that specifies all its variable parameters. The driver also contains two simple declaration macros that simplify the full declaration by default object values, but the count of bitmap is restricted to two bitmaps.

The main icon object definition macro is created from three individual parts that allows to indicate the complete icon with various counts of bitmaps:

- Begin part—This part specifies all the necessary parameters about the icon object itself.
- Add bitmaps part—This part allows multiple D4D_DECLARE_ICON_BMP macros to be used to add all icon bitmaps.
- End part—This part is used only to close the icon object bitmaps array definition

D4D_DECLARE_ICON_BEGIN(name, text, x, y, cx, cy, tx, ty, flags, pScheme, fontId, pUser, pOnValch, pOnUsrMsg)

Input parameters:

- name—Name of an icon object
- text—Title text of an icon
- x—Position of an icon in the client area of the screen in axis X
- y—Position of an icon in the client area of the screen in axis Y
- cx—Size of an icon in the axis X
- cy—Size of an icon in the axis Y
- tx—Offset of title in axis X
- ty—Offset of title in axis Y
- flags—Bit mask that specifies the initial system object flags and icon object flags
- pScheme—Pointer to a color scheme, if it is NULL, the default scheme is used

- fontId—Identification number of the used font
- pUser—Pointer on void, can be used by user application any way the user wants
- onvalch—Pointer to the on-value change user callback function (in format: “void (*OnValueChanged)(D4D_OBJECT_PTR pThis)”).
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

Description—This macro describes all icon object parameters except a bitmap array definition.

D4D_DECLARE_ICON_BMP(pBmp)

Input parameters—pBmp—Pointer to a bitmap

Description—This macro is used to add a bitmap to the icon object bitmaps array. It can be used multiple times.

D4D_DECLARE_ICON_END()

Input parameters—NA

Description—This macro is used to close the icon object bitmaps array. It must be placed after the last bitmap array declaration macro.

D4D_DECLARE_STD_ICON_BEGIN(name, text, x, y, cx, cy, tx, ty, fontId, onvalch)

Input parameters—All parameters in the declaration have the same purpose as in a full declaration macro, the remaining parameters have default values:

- flags—Has the value of macro D4D_ICON_F_DEFAULT
- pScheme—Has NULL value, the default color scheme will be used
- pUser—Has NULL value
- pOnUsrMsg—Has NULL value

Description—This is the default icon definition. The main advantage is less parameters of this macro against the full version.

D4D_DECLARE_STD_ICON_BEGIN_AUTOSIZE(name, text, x, y, fontId, onvalch)

Input parameters—All parameters in the declaration have the same purpose as in the standard declaration macro, the rest of the parameters (cx, cy, tx, ty) are set to zero to enable auto size capability.

Description—This is a default icon definition with enabled auto size capability. The main advantage is less parameters of the icon declaration.

D4D_DECLARE_ICON_BEGIN_AUTOSIZE(name, text, x, y, flags, pScheme, fontId, pUser, onvalch, pOnUsrMsg)

Input parameters—All parameters in the declaration have the same purpose as in a full declaration macro, the rest of the parameters (cx, cy, tx, ty) are set to zero to enable auto size capability.

Description—This is an icon definition with enabled auto size capability. The main advantage of this macro is missing four size parameters.

D4D_DECLARE_STD_ICON1(name, text, x, y, cx, cy, tx, ty, pBmp0, font, onvalch)

Input parameters—All parameters in the declaration have the same purpose as in a standard declaration macro and the added pBmp0 parameter is a pointer to a bitmap.

Description—This is a simplified default icon definition for an icon with only one bitmap. The main advantage is only one declaration macro that specifies a complete icon object instead of standard three declaration macros.

D4D_DECLARE_STD_ICON2(name, text, x, y, cx, cy, tx, ty, pBmp1, pBmp0, font, onvalch)

Input parameters—All parameters in the declaration have the same purpose as in a standard declaration macro. The added pBmp0 and pBmp1 parameters are pointers to a bitmap.

Description—This is a simplified default icon definition for an icon with only two bitmaps. The main advantage is one declaration macro that specifies a complete icon object instead of a standard three declaration macros.

3.2.9.5.2 D4D_ICON Predefined Standard Constants

The icon object contains a few predefined constants that are used in a standard icon declaration. All of these constants can be modified in the user configuration file.

The screen behavior and visual aspect flags:

- D4D_ICON_F_DEFAULT — This is a help macro used for default configuration (value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED).
- D4D_ICON_TXT_PRTY_DEFAULT — This is a text property macro used for default configuration of gauge text (value: D4D_TXT_PRTY_ALIGN_H_CENTER_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).
- D4D_ICON_FNT_PRTY_DEFAULT — This is a font property macro used for default configuration of gauge font (value: 0).

3.2.9.5.3 D4D_ICON Functions

void D4D_IconSetIndex(D4D_OBJECT_PTR pThis, D4D_ICON_INDEX index);

Input parameters:

- pThis—Pointer to an icon object
- index—Index of a new bitmap that must be shown

Output parameters—NA

Description—The function is used to change the shown bitmap of an icon object to any other in the bitmap array by the index parameter.

D4D_ICON_INDEX D4D_IconGetIndex(D4D_OBJECT_PTR pThis);

Input parameters—pThis—Pointer to an icon object

Output parameters—Index of the shown bitmap

Description—The function is used to get the index of the shown bitmap of an icon object.

void D4D_IconChangeIndex(D4D_OBJECT_PTR pThis, sByte incr);

Input parameters:

- pThis—Pointer to an icon object.
- incr—Signed increment value that specifies the relative new value of the shown icon index to the current one.

Output parameters—NA

Description—The function is used to change the shown bitmap of an icon object to any other in the bitmap array by a signed increment parameter.

void D4D_IconSetText(D4D_OBJECT_PTR pObj, char* pText);

Input parameters:

- pObj—Pointer to an icon object
- pText—Pointer to a new string

Output parameters—NA

Description—Function changes the title text of the icon. To have success running this function, the original declared text in the instantiation macro has to be placed in RAM.

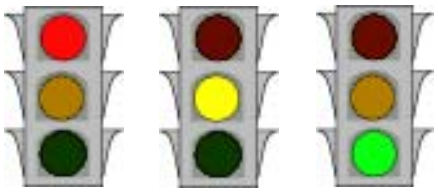
NOTE

This function is not preferred because version 1.0 and the direct replacement is a general function D4D_SetText with the same parameters and behavior.

3.2.9.5.4 Example of Use D4D_ICON

```
static void OnChange_Icon1(D4D_OBJECT* pThis);

D4D_DECLARE_BMP(scr1_bmpIconRed, icnRedBmp, NULL)
D4D_DECLARE_BMP(scr1_bmpIconYellow, icnYellowBmp, NULL)
D4D_DECLARE_BMP(scr1_bmpIconGreen, icnGreenBmp, NULL)
```



```
D4D_DECLARE_STD_ICON_BEGIN_AUTOSIZE(scr1_semaphore_Icon, NULL, 60, 80, FONT_8x14_BIG,
OnChange_Icon1)
    D4D_DECLARE_ICON_BMP(&scr1_bmpIconRed)
    D4D_DECLARE_ICON_BMP(&scr1_bmpIconYellow)
    D4D_DECLARE_ICON_BMP(&scr1_bmpIconGreen)
D4D_DECLARE_ICON_END()

D4D_DECLARE_SCREEN_BEGIN(screenicon, ScreenIcon_)
```

```
D4D_DECLARE_SCREEN_OBJECT(scr1_semaphore_Icon)
D4D_DECLARE_SCREEN_END()
```

```
static void OnChange_Icon1 (D4D_OBJECT* pThis)
{
    // Does something when the icon value is changed
}
```

3.2.9.6 D4D_SLIDER

The D4D_SLIDER object is prepared to be used as a visualization and control object to show and set the numerical values of the application in a graphical form and provide many basic features:

- Variable position, size, bar offset, bar size, and text position
- Title text that can be run-time changed
- Underline, strike through and transparent text option
- Disable and enable capability
- Background bitmap
- Touch screen support
- Variable colors of the text, bar, and background
- Automatic color cross function from minimum to maximum value of the slider
- OnValueChange callback function to simplify the user application
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

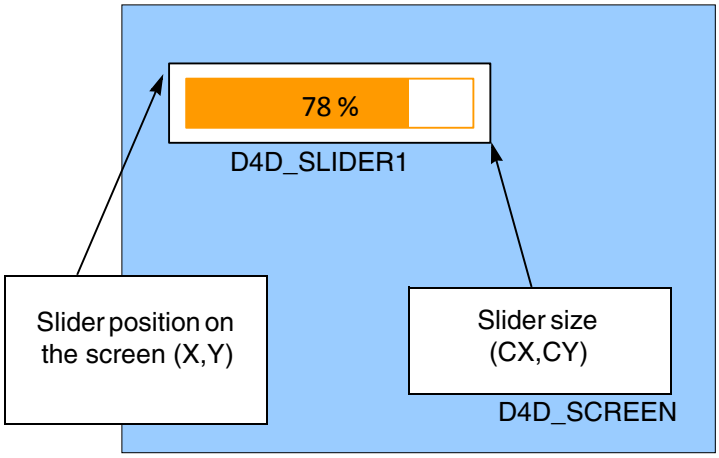


Figure 3-16. Slider location and description

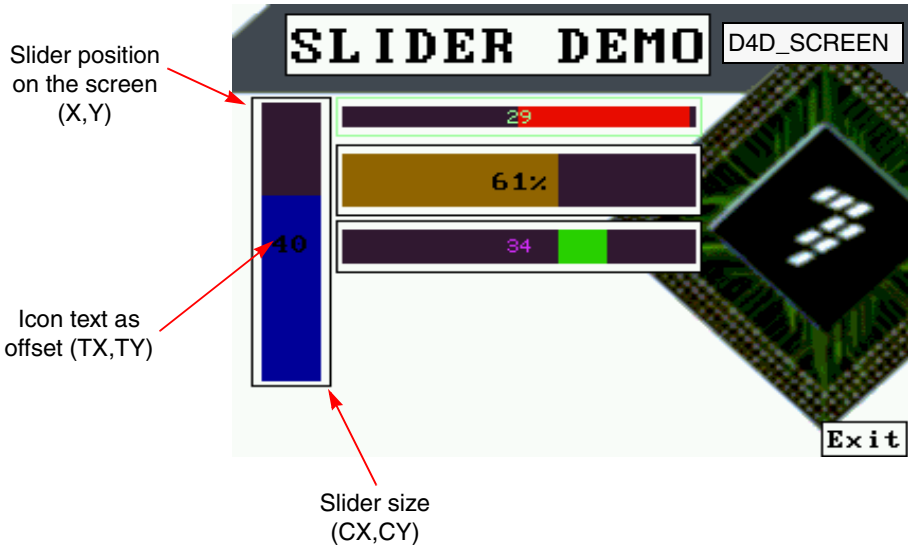


Figure 3-17. Slider location and description example

3.2.9.6.1 D4D_SLIDER Instantiation Macros

D4D_DECLARE_SLIDER(name, text, x, y, cx, cy, tx, ty, bx, by, bcx, bcy, flags, pBmp, pScheme, fontId, pUser, onvalch, pOnUsrMsg)

Input parameters:

- name—Name of a slider object
- text—Title text of a slider
- x—Position of a slider in the client area of the screen in axis X
- y—Position of a slider in the client area of the screen in axis Y
- cx—Size of a slider in axis X

- cy—Size of a slider in axis Y
- tx—Offset of a title in axis X
- ty—Offset of a title in axis Y
- bx—Offset of a bar in axis X
- by—Offset of a bar in axis Y
- bcx—Size of a slider bar in axis X
- bcy—Size of a slider bar in axis Y
- flags—Bitmask that specifies initial system object flags and gauge object flags
- pbmp—Pointer to a bitmap that is shown in the background of a slider
- pScheme—Pointer to a color scheme, if it is NULL, the default scheme is used
- fontId—Identification number of the used font of the text
- pUser—Pointer on void, that can be used by user application any way the user wants
- onvalch—Pointer to the on-value change user callback function (in format: “void (*OnValueChanged)(D4D_OBJECT_PTR pThis)”).
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

Description—This macro creates a complete slider object by individually setting all parameters of this object.

D4D_DECLARE_STD_SLIDER(name, text, x, y, cx, cy, tx, ty, bx, by, bcx, bcy, pBmp, fontId, onvalch)

Input parameters—All parameters in the declaration have the same purpose as in a full declaration macro, the remaining of the parameters have default values:

- flags—Has value of macro D4D_SLDR_F_DEFAULT.
- pScheme—Has NULL value, the default color scheme will be used.
- pUser—Has NULL value.
- pOnUsrMsg—Has NULL value.

Description—This is the default slider definition. The main advantage is less parameters of this macro against the full version.

D4D_DECLARE_STD_SLIDER_AUTOSIZE(name, text, x, y, cx, cy, pBmp, font, onvalch)

Input parameter—All parameters in the declaration have the same purpose as in a standard declaration macro. The remaining parameters (tx, ty, bx, by, bcx, bcy) are set to zero to enable auto size capability:

Description—This is the default slider definition with auto size capability enabled. The main advantage is less parameters of the slider declaration.

D4D_DECLARE_SLIDER_AUTOSIZE(name, text, x, y, cx, cy, flags, pBmp, pScheme, font, pUser, onvalch, pOnUsrMsg)

Input parameters—All parameters in the declaration have the same purpose as in a full declaration macro. The rest of the parameters (tx, ty, bx, by, bcx, bcy) are set to zero to enable the auto size capability

Description—This is a slider definition with enabled auto size capability. The main advantage of this macro is missing six size and offset parameters.

3.2.9.6.2 D4D_SLIDER Predefined Standard Constants

The slider object contains a few predefined constants that are used in a standard slider declaration. All of these constants can be modified in the user configuration file.

The color definition:

The slider bar color:

- D4D_COLOR_SLDRBAR (default value: D4D_COLOR_DARK_BLUE)

The slider bar background color:

- D4D_COLOR_SLDRBAR_BKGD (default value: D4D_COLOR_WHITE)

The slider bar start color:

- D4D_SLDR_CLR_START_DEFAULT (default value: D4D_COLOR_GREEN)

The slider bar end color:

- D4D_SLDR_CLR_END_DEFAULT (default value: D4D_COLOR_RED)

The coordination definition:

The default offset of the slider bar length:

- D4D_SLDR_BAR_OFF_LENGTH (default value: 2)

The default offset of the slider bar width:

- D4D_SLDR_BAR_OFF_WIDTH (default value: 4)

The screen behavior and visual aspect flags:

- D4D_SLDR_F_BAR_AUTOCOLOR—This flag specifies the behavior of the slider bar. If it is set, the color of a slider bar depends on the current value of the slider and it is computed from a color scheme object dependent parameters barStart and barEnd from the slider declaration macro. In another situation, a color specified by the bar parameter from the color scheme object dependent parameters are used.
- D4D_SLDR_F_TEXT_AUTOCOLOR—This flag specifies the behavior of the slider text. When set, then the inverse color of the slider bar color is used for the slider text. In another case color is constant with the value of the standard forecolor of the used color scheme.
- D4D_SLDR_F_DEFAULT—This is a help macro that is used for default configuration (value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED | D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FOCUSRECT | D4D_OBJECT_F_TRANSP_TEXT).

- D4D_SLDR_TXT_PRTY_DEFAULT — This is a text property macro used for default configuration of slider text (value: D4D_TXT_PRTY_ALIGN_H_CENTER_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).
- D4D_SLDR_FNT_PRTY_DEFAULT — This is a font property macro used for default configuration of slider font (value: D4D_FNT_PRTY_TRANSPARENT_YES_MASK).

3.2.9.6.3 D4D_SLIDER Functions

void D4D_SldrSetValue(D4D_OBJECT_PTR pThis, D4D_SLIDER_VALUE value);

Input parameters:

- pThis—Pointer to a slider object
- value—A new value of the slider, must fit into the range set in D4D_SLIDER_LIMITS

Output parameters—NA

Description—The function sets the new value of a slider

D4D_SLIDER_VALUE D4D_SldrGetValue(D4D_OBJECT_PTR pThis);

Input parameters—pThis—Pointer to a slider object

Output parameters—Current set value of the slider

Description—Function returns the current value of a slider object

void D4D_SldrChangeValue(D4D_OBJECT_PTR pThis, D4D_SLIDER_VALUE incr);

Input parameters:

- pThis—Pointer to a slider object.
- incr—Signed increment value that specifies a relative new value of the slider to the current one.

Output parameters—NA

Description—This function is used to change a slider value to a new one by incrementing the parameter.

void D4D_SldrSetLimits(D4D_OBJECT_PTR pThis, const D4D_SLIDER_LIMITS* pLimits);

Input parameters:

- pThis—Pointer to a slider object
- pLimits—Pointer to a new D4D_SLIDERR_LIMITS structure that is used as new in a slider object

Output parameters—NA

Description—The function loads a new limits value to the slider object and also takes care of adapting the current value of the slider to fit into a new limits range.

void D4D_SldrGetLimits(D4D_OBJECT_PTR pThis, D4D_SLIDER_LIMITS* pLimits);

Input parameters:

- pThis—Pointer to a slider object

- pLimits—Pointer to the D4D_SLIDER_LIMITS structure in a volatile memory used to store current limit value of the slider object.

Output parameters—NA

Description—This function copies the current set limits of a slider object to the D4D_SLIDER_LIMITS structure in a volatile memory.

void D4D_SldrSetBarColor(D4D_OBJECT_PTR pThis, D4D_COLOR color);

Input parameters:

- pThis—Pointer to a slider object.
- color—A new value of color of a slider bar.

Output parameters—NA

Description—The function sets the new color of the slider. It can be used only when the auto color bar option is disabled.

D4D_COLOR D4D_SldrGetBarColor(D4D_OBJECT_PTR pThis);

Input parameters— pThis—Pointer to a slider object

Output parameter—Current used slider bar color

Description—The function returns the current color of a slider bar

void D4D_SldrSetText(D4D_OBJECT_PTR pObj, char* pText);

Input parameters:

- pObj—Pointer to a slider object
- pText—Pointer to a new string

Output parameters—NA

Description—Function changes the title text of the slider. To have success running this function, the original declared text in the instantiation macro has to be placed in RAM.

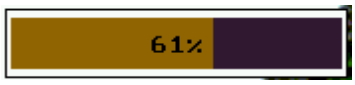
NOTE

This function is not preferred because version 1.0 and direct replacement is the general function D4D_SetText with the same parameters and behavior.

3.2.9.6.4 Example of Use D4D_SLIDER

```
static void OnChange_Slider1(D4D_OBJECT* pThis);

D4D_DECLARE_STD_SLIDER(scr1_slider1, “61%”, 10, 50, 35, 150, 8, 0, 0, 0, 0, 0, NULL, FONT_8x14,
OnChange_Slider1)
D4D_DECLARE_SCREEN_BEGIN(screenslider, ScreenSlider_)
D4D_DECLARE_SCREEN_OBJECT(scr1_slider1)
D4D_DECLARE_SCREEN_END()
```




```
static void OnChange_Slider1 (D4D_OBJECT* pThis)
{
    // Does something when the slider value is changed
}
```

3.2.9.7 D4D_MENU

The D4D_MENU object is prepared to be used to create a simple menu with the menu title and a side bar that provides a lot of basic features:

- Variable position and size
- Title header with an icon and optional index of selected menu item
- Optional side bar for easy orientation in a bigger menu
- Auto size capability for size and title text offset
- Disable and enable capability
- Underline and strike through text option
- Touch screen support to focus and change action with fast touch capability (both actions in one touch)
- Variable colors of text and background in a normal and focus state
- OnChange callback function to simplify the user application
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

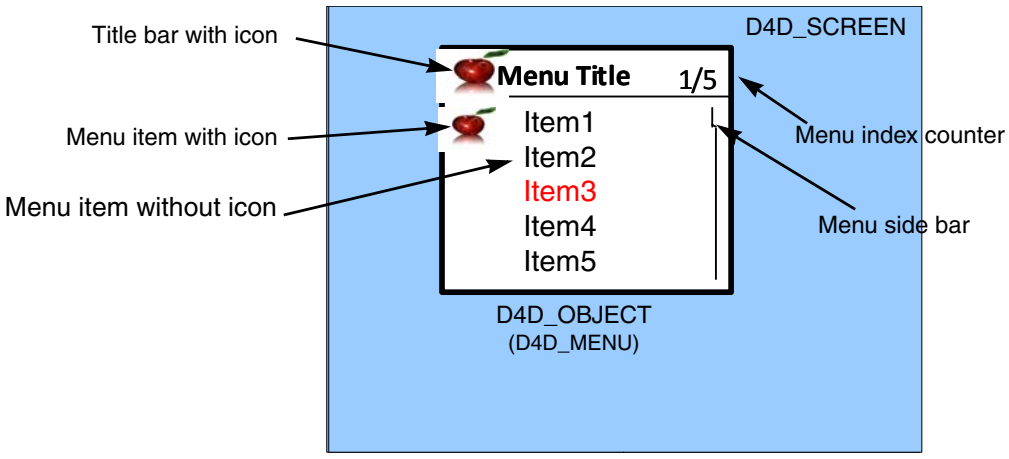


Figure 3-18. MENU object on the screen

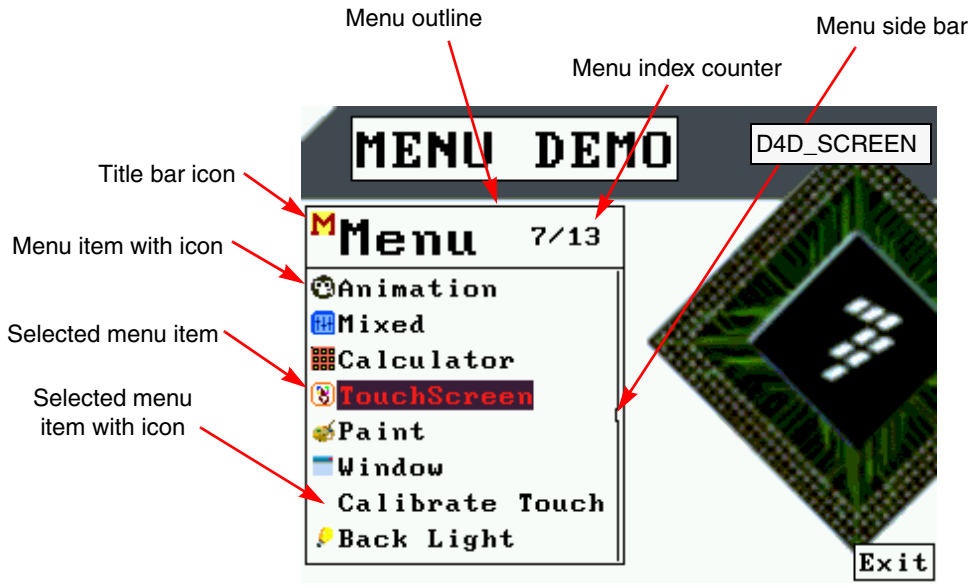


Figure 3-19. MENU object on the screen example

3.2.9.7.1 D4D_MENU Instantiation Macros

The menu object has an instantiation macro in the same shape as the screen declaration macro that specifies all its variable parameters. The driver also contains two simpler declaration macros that simplify full declaration by the default object values.

The menu object definition macro is created from three individual parts that allow to specify a complete menu object with various counts of menu items:

- Begin part—This part specifies all necessary parameters of the menu object itself.
- Add menu items part—This part allows to use multiple D4D_DECLARE_MENU_ITEM macros to add all menu items.
- End part—This part is used only to close the menu object items array definition.

D4D_DECLARE_MENU_BEGIN(name, title_text, title_font, x, y, cx, cy, flags, pScheme, IndexFontId, ItemsFontId, posCnt, MenuItemsOff, pIcon, pUser, pOnClick, pOnUsrMsg)

Input parameters:

- name—Name of the menu object
- title_text—Title text of the menu object
- title_font—Identification number of the font used in the title
- x—Position of a menu in the client area of the screen in axis X
- y—Position of a menu in the client area of the screen in axis Y
- cx—Size of the menu in axis X
- cy—Size of the menu in axis Y
- flags—Bitmask that specifies the initial system object flags and menu object flags

- pScheme—Pointer to a color scheme, if it is NULL, the default scheme is used
- ItemsFontId—Identification number of the used font in the items text
- PosCnt—The count of shown items in the menu object, when it is set to zero, it is computed automatically
- MenuItemsOff—Vertical offset in pixels between two menu items, when it is set to zero, it is computed automatically
- pIcon—Pointer to a bitmap that is used as a menu title icon
- pUser—Pointer on void, that can be used by the user application any way the user wants
- IndexFontId—Identification number of the used font of the index counter text
- onclick—Pointer to the on-click user callback function (in format: “void (*OnClick)(D4D_OBJECT* pThis, D4D_MENU_INDEX ix)”).
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

Description—This is a full definition macro that allows setting all parameters of the menu object individually, it allows modifying all options of the menu object.

D4D_DECLARE_MENU_ITEM(text, pIcon)

Input parameters: text—Text of menu item

pIcon—Pointer to a bitmap that is used as a menu item icon

Description—This macro is use to add an item to the menu object items array. It can be used multiple times.

D4D_DECLARE_MENU_END(name)

Input parameters: name—The name has to be the same as name parameter in the D4D_DECLARE_MENU_BEGIN declaration.

Description—This macro is used to close a menu object items array. It must be placed after the last item array declaration macro.

D4D_DECLARE_STD_MENU_BEGIN(name, title_text, title_font, x, y, cx, cy, IndexFontId, ItemsFontId, posCnt, MenuItemsOff, pIcon, pOnClick)

Input parameters—All parameters in the declaration have the same purpose as in a full declaration macro, rest of the parameters has default values:

- flags—Has a value of macro D4D_MENU_F_DEFAULT
- pScheme—Has NULL value, the default color scheme will be used
- pUser—Has NULL value
- pOnUsrMsg—Has NULL value

Description—This is the default menu definition. The main advantage is less parameters of this macro against the full version.

D4D_DECLARE_STD_MENU_AUTOSIZE_BEGIN(name, title_text, title_font, x, y, cx, cy, IndexFontId, ItemsFontId, pIcon, pOnClick)

Input parameters—Input parameters are the same as in a standard menu definition, only posCnt and MenuItemsOff parameters are replaced by zero that invoke auto size capability for these parameters.

Description—This is a definition that has as few parameters as possible to configure a standard menu object. The size values are generated run-time from size of a used item font and title text font.

3.2.9.7.2 D4D_MENU Predefined Standard Constants

The menu object contains a few predefined constants that are used in a standard menu declaration and helps to indicate the sizes resulting when the auto size is enabled. All of these constants can be modified in the user configuration file.

The screen behavior and visual aspect flags:

The menu index counter enables flag:

- D4D_MENU_F_INDEX—This bit enables the menu index counter

The menu side bar enables the flag:

- D4D_MENU_F_SIDEBAR—This bit enables the menu side bar

The menu default flags:

- D4D_MENU_F_DEFAULT—This is a help macro that is used for default configuration (value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED | D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FOCUSRECT | D4D_MENU_F_INDEX | D4D_MENU_F_SIDEBAR).
- D4D_MENU_TXT_PRTY_DEFAULT—This is a text property macro used for default configuration of the menu title text (value: D4D_TXT_PRTY_ALIGN_H_CENTER_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).
- D4D_MENU_FNT_PRTY_DEFAULT—This is a font property macro used for default configuration of the menu title font (value: 0).
- D4D_MENU_IX_TXT_PRTY_DEFAULT—This is a text property macro used for default configuration of menu index text (value: D4D_TXT_PRTY_ALIGN_H_CENTER_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).
- D4D_MENU_IX_FNT_PRTY_DEFAULT—This is a font property macro used for default configuration of the menu index font (value: 0).
- D4D_MENU_ITEM_TXT_PRTY_DEFAULT—This is a text property macro used for default configuration of menu items text (value: D4D_TXT_PRTY_ALIGN_H_LEFT_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).
- D4D_MENU_ITEM_FNT_PRTY_DEFAULT—This is a font property macro used for default configuration of the menu items font (value: 0).

3.2.9.7.3 D4D_MENU Functions

D4D_MENU_INDEX D4D_MenuGetIndex(D4D_OBJECT* pThis);

Input parameters:

- pThis—Pointer to a menu object

Output parameters—Index of current selected menu item

Description—The function returns the current index of the selected item

3.2.9.7.4 Example of Use D4D_MENU

```
static void OnClick_Menu1(D4D_OBJECT* pThis, D4D_MENU_INDEX ix);

D4D_DECLARE_BMP(scrmenu_Menu, icon_menu12x12, NULL)
```

```
D4D_DECLARE_BMP(scrmenu_AnimIcon, icon_animated12x12, NULL)
```

```
D4D_DECLARE_BMP(scrmenu_Mix, icon_mix12x12, NULL)
```

```
D4D_DECLARE_BMP(scrmenu_Gauge, icon_gauge12x12, NULL)
```

```
D4D_DECLARE_BMP(scrmenu_Slider, icon_slider12x12, NULL)
```

```
D4D_DECLARE_BMP(scrmenu_Icon, icon_icon12x12, NULL)
```

```
D4D_DECLARE_BMP(scrmenu_KeyPad, icon_keypad_12x12, NULL)
```

```
D4D_DECLARE_BMP(scrmenu_Touch, icon_Touchscreen12x12, NULL)
```

```
D4D_DECLARE_BMP(scrmenu_Paint, icon_paint12x12, NULL)
```

```
D4D_DECLARE_BMP(scrmenu_Window, icon_window12x12, NULL)
```



```
D4D_DECLARE_BMP(scrmenu_Bulb, icon_bulb12x12, NULL)
```



```
D4D_DECLARE_STD_MENU_AUTOSIZE_BEGIN(scr1_Menu1, "Menu", FONT_8x14_BIG, 2, 50, 160, 180,
FONT_8x14, FONT_8x14, &scrmenu_Menu, OnClick_Menu1)

D4D_DECLARE_MENU_ITEM("Gauge", &scrmenu_Gauge)
D4D_DECLARE_MENU_ITEM("Slider", &scrmenu_Slider)
```

```
D4D_DECLARE_MENU_ITEM("Icon", &scrmenu_Icon)
D4D_DECLARE_MENU_ITEM("Animation", &scrmenu_AnimIcon)
D4D_DECLARE_MENU_ITEM("Mixed", &scrmenu_Mix)
D4D_DECLARE_MENU_ITEM("Calculator", &scrmenu_KeyPad)
D4D_DECLARE_MENU_ITEM("TouchScreen", &scrmenu_Touch)
D4D_DECLARE_MENU_ITEM("Paint", &scrmenu_Paint)
D4D_DECLARE_MENU_ITEM("Window", &scrmenu_Window)
D4D_DECLARE_MENU_ITEM("Calibrate Touch", NULL)
D4D_DECLARE_MENU_ITEM("Back Light", &scrmenu_Bulb)
D4D_DECLARE_MENU_ITEM("Screen Example", NULL)
D4D_DECLARE_MENU_END(scr_Menu_Menu)

D4D_DECLARE_SCREEN_BEGIN(screenmenu, ScreenMenu_)
D4D_DECLARE_SCREEN_OBJECT(scr1_Menu1)
D4D_DECLARE_SCREEN_END()
```


```
static void OnClick_Menu1(D4D_OBJECT* pThis, D4D_MENU_INDEX ix)
{
    // Does something when the is clicked on any menu item
}
```

3.2.9.8 D4D_LABEL

The D4D_LABEL object is prepared to be used as a visualization object to show simple text one-line information of an application in a graphical form and provides a few basic features:

- Variable position and size
- Text that can be run-time changed
- Underline, strike through and transparent text option
- Disable/enable capability
- Text align—Left, center, top, bottom, and right
- Touch screen support to focus
- Variable colors of text and background
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

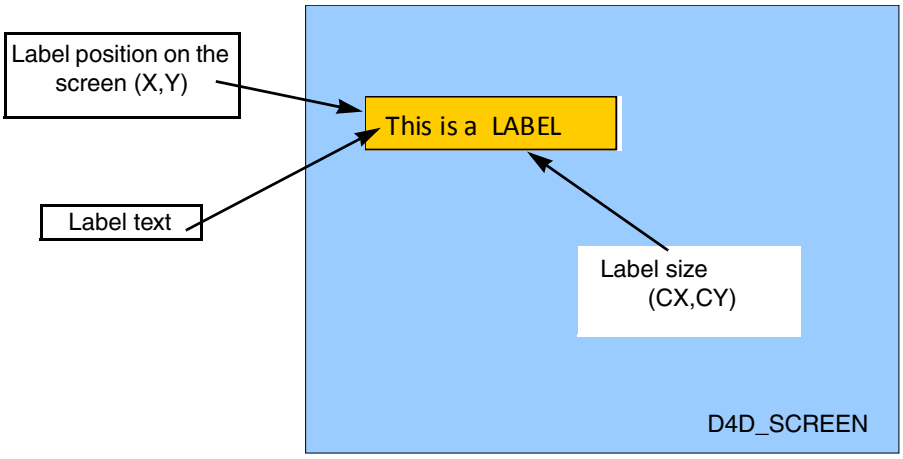


Figure 3-20. Label object

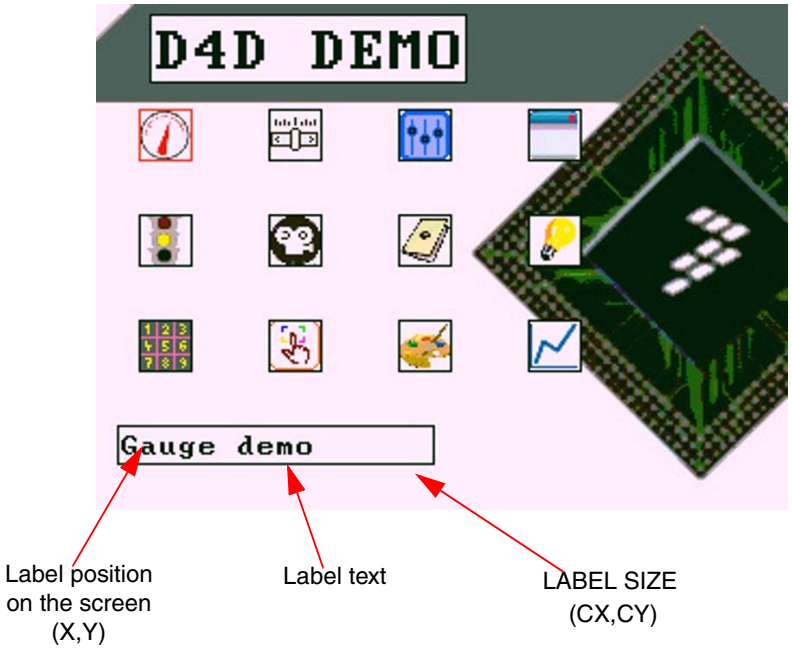


Figure 3-21. Label object example

3.2.9.8.1 D4D_LABEL Instantiation Macros

D4D_DECLARE_LABEL(name, text, x, y, cx, cy, flags, pScheme, fontId, pUser, pOnUsrMsg)

Input parameters:

- name—Name of a label object
- text—Text of a label
- x—Position of a label in the client area of the screen in axis X
- y—Position of a label in the client area of the screen in axis Y
- cx—Size of a label in axis X
- cy—Size of a label in axis Y
- flags—Bitmask that specifies the initial system object flags and label object flags
- pScheme—Pointer to a color scheme, if it is NULL, the default scheme is used
- fontId—Identification number of the used font of the text
- pUser—Pointer on void, can be used by the user application any way the user wants
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

Description—This macro creates a complete label object with individually setting all parameters of this object.

D4D_DECLARE_STD_LABEL(name, text, x, y, cx, cy, font)

Input parameters—All parameters in the declaration have the same purpose as in full declaration macro, the rest of the parameters have default values:

- flags—Has the value of the macro D4D_LBL_F_DEFAULT.
- pScheme—Has NULL value, the default color scheme will be used.
- pUser—Has NULL value.
- pOnUsrMsg—Has NULL value.

Description—This is a default label definition. The main advantage is less parameters of this macro against one full version.

D4D_DECLARE_STD_LABEL_AUTOSIZE(name, text, x, y, font)

Input parameters—All parameters in the declaration have the same sense as in a standard declaration macro, the remaining parameters (cx, cy) are set to zero to enable the auto size capability.

Description—This is a default label definition with enabled auto size capability. The main advantage is less parameters of the slider label declaration.

D4D_DECLARE_LABEL_AUTOSIZE(name, text, x, y, flags, pScheme, font, pUser, pOnUsrMsg)

Input parameters—All parameters in the declaration have the same purpose as in a full declaration macro, the remaining parameters (cx, cy) are set to zero to enable the auto size capability.

Description—This is a label definition with enabled auto size capability. The main advantage of this macro is that it is missing two size parameters.

3.2.9.8.2 D4D_LABEL Predefined Standard Constants

The slider object contains a few predefined constants that are used in the standard slider declaration. All of these constants can be modified in the user configuration file.

The screen behavior and visual aspect flags:

- D4D_LBL_F_DEFAULT—This is a help macro that is used for default configuration (value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED | D4D_OBJECT_F_FOCUSRECT).
- D4D_LBL_TXT_PRTY_DEFAULT—This is a text property macro used for default configuration of label text (value: D4D_TXT_PRTY_ALIGN_H_CENTER_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).
- D4D_LBL_FNT_PRTY_DEFAULT—This is a font property macro used for default configuration of the label font (value: 0).

3.2.9.8.3 D4D_LABEL Functions

void D4D_LabelSetText(D4D_OBJECT_PTR pObj, char* pText)

Input parameters:

- pObj—Pointer to a label object
- pText—Pointer to a new string

Output parameters—NA

Description—Function changes the title text of the slider. To have success running this function, the original declared text in an instantiation macro has to be placed in RAM.

NOTE

This function is not preferred because version 1.0 and direct the replacement is the general function D4D_SetText with the same parameters and behavior.

3.2.9.8.4 Example of Use D4D_LABE

```
D4D_DECLARE_STD_LABEL_AUTOSIZE(scr1_label1, “MENU DEMO”, 25, 5, FONT_8x14_BIG)

D4D_DECLARE_SCREEN_BEGIN(screenlabel, ScreenLabel_)
D4D_DECLARE_SCREEN_OBJECT(scr1_label1)
D4D_DECLARE_SCREEN_END()
```

3.2.9.9 D4D_GRAPH

The D4D_GRAPH object is prepared to create a simple graph that provides many basic features:

- Variable position and size
- Title header of the graph
- Selectable grid of graph in both axes
- X-axis scale capability
- Labels for both axes
- Auto size capability for the size of the graph (not the whole object)
- Disable/enable capability
- Multiple traces of the graph
- Variable colors and styles of traces
- OnChange callback function to simplify the user application
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped with the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

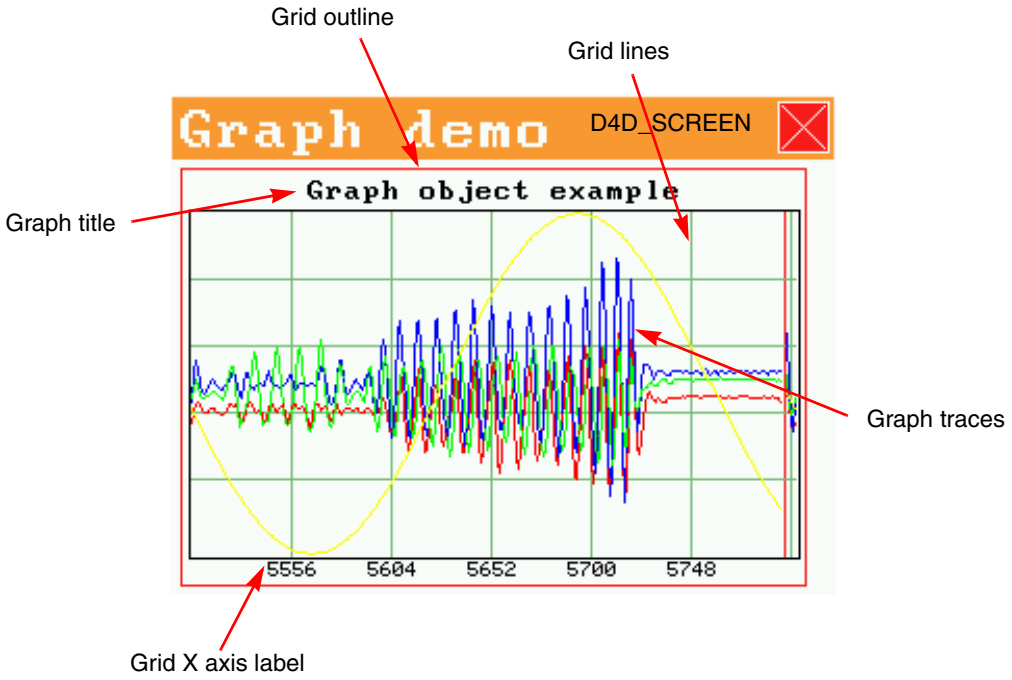


Figure 3-22. Graph object on the screen example

3.2.9.9.1 D4D_GRAPH Instantiation Macros

The graph object has an instantiation macro in the same shape as the screen or menu declaration macro that specifies all its variable parameters. The driver also contains a simpler declaration macros that simplify the full declaration by default object values.

The graph object definition macro is created from three individual parts that allow to indicate a complete graph object with various counts of graph traces:

- Begin part—This part specifies all the necessary parameters of the graph object itself.
- Add trace to graph—This part allows to use multiple D4D_DECLARE_GRAPH_TRACE macros to add all traces into the graph.
- End part—This part is used only to close the graph traces array definition.

D4D_DECLARE_GRAPH_BEGIN(name, text, x, y, cx, cy, gx, gy, dataLen, flags, pScheme, fontId, lblFontId, pOnNeedLblTxt, pUser, pOnValch, pOnUsrMsg)

Input parameters:

- name—Name of a graph object
- text—Title text of a graph object
- x—Position of a graph in the client area of the screen in axis X
- y—Position of a graph in the client area of the screen in axis Y
- cx—Size of the graph in axis X
- cy—Size of the graph in axis Y
- gx—Count of grid lines in axis X
- gy—Count of grid lines in axis Y
- dataLen—Length of data buffers for traces
- flags—Bitmask that specifies the initial system object flags and graph object flags
- pScheme—Pointer to a color scheme, if it is NULL, the default scheme is used
- FontId—Identification number of the used font for the title
- lblFontId—Identification number of the used font for the axis labels
- ,pOnNeedLblTxt—Pointer to the function that is called when text is needed to draw a label (in format: “char* (*OnNeedLabelText)(D4D_OBJECT_PTR pThis, D4D_BOOL axisX, D4D_INDEX gridIx)”).
- pUser—user pointer—Free pointer on void, that can be used by the user application any way the user wants.
- pOnValch—Pointer to the on-value changed user callback function (in format: “void (*OnValueChanged)(D4D_OBJECT_PTR pThis)”).
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

Description—This is a full definition macro that allows setting all parameters of the graph object individually. It allows modifying all options of a graph object.

D4D_DECLARE_GRAPH_TRACE(pData, clr, line, type)

Input parameters—pData—Pointer to a trace input buffer.

- clr—Color of a trace
- line—Line type of a trace
- type—Type of a trace

Description—This macro is used to add a trace to graph object traces array. It can be used multiple times.

D4D_DECLARE_GRAPH_END()

Input parameters—None

Description—This macro is used to close a graph object traces array. It must be placed after the last trace array declaration macro.

D4D_DECLARE_STD_GRAPH_BEGIN(name, text, x, y, cx, cy, gx, gy, dataLen, fontId, valFontId)

Input parameters—All parameters in the declaration have the same purpose as in a full declaration macro, the remaining of the parameters have the default values:

- flags—Has value of macro D4D_GRAPH_F_DEFAULT
- pScheme—Has NULL value, the default color scheme will be used
- pUser—Has NULL value
- pOnUsrMsg—Has NULL value

Description—This is a default graph definition. The main advantage is less parameters of this macro against the full version.

3.2.9.9.2 D4D_GRAPH Types

The D4D_GRAPH object has any own types that it needs to run.

List of Graph object data types:

- D4D_GRAPH_VALUE—The type of input graph data. Type is unsigned char.
- D4D_GRAPH_VALUE_LEN—The length of the graph. This type restricts the maximum size of input data buffer and maximum of samples shown on the screen by scale function.Type is unsigned short.
- D4D_GRAPH_SAMPLE_IX—The of type for index of input samples. This type restricts the maximum size of all samples. Type is unsigned int.

3.2.9.9.3 D4D_GRAPH Predefined Standard Constants

The graph object contains a few predefined constants that are used in standard graph declaration. Modifiable constants could be changed in the user configuration file.

The graph standard color:

- D4D_COLOR_GRAPH_GRID—This macro contains a standard color for grid lines in the graph (default value is: D4D_COLOR_GREY).

The graph coordination and size definition:

- D4D_GRAPH_BORDER_OFF—This macro specifies the dimension of the offset between the main object outline and the graph itself (default value is: 5 pixels).
- D4D_GRAPH_VALUE_OFF—This macro specifies the dimension of axis labels offsets between the graph itself and axis labels (default value is: 2 pixels).

The screen behavior and visual aspect flags:

The graph type:

- D4D_GRAPH_F_MODE_NORMAL—The graph runs in normal mode, this means that a graph adds new data to the screen. When it fills up the whole graph area, the object clears all the shown points and draws a new graph on a screen from new data.
- D4D_GRAPH_F_MODE_ROLLOVER—The graph runs in roll-over mode, this means the graph adds new data to the screen. When it fills up the whole graph area, the graph clears only the latest sample and draws a new one instead of the cleared one.

The labels types:

- D4D_GRAPH_F_VALUE_X_BOTT—Enables a label for axis X on the bottom side of the graph.
- D4D_GRAPH_F_VALUE_X_TOP—Enables the label for axis X on the top side of the graph.
- D4D_GRAPH_F_VALUE_Y_LEFT—Enables the label for axis Y on the left side of the graph.
- D4D_GRAPH_F_VALUE_Y_RIGHT—Enables the label for axis Y on the right side of the graph.

NOTE

For axis, only one setting can be used, it is not allowed to mix two flags for one axis.

The trace type:

- D4D_GRAPH_TRACE_TYPE_LINE—The trace looks like a normal line, all individual sample points are connected by the lines.
- D4D_GRAPH_TRACE_TYPE_POINT—The trace is drawn only from measured points.
- D4D_GRAPH_TRACE_TYPE_AREA—The trace fills up the area under measured samples.
- D4D_GRAPH_TRACE_TYPE_AREA_INV—The trace fills up the area of measured samples.

The graph default flags:

- D4D_GRAPH_F_DEFAULT—This is a help macro that is used for default configuration (value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED | D4D_OBJECT_F_FOCUSRECT | D4D_GRAPH_F_MODE_ROLLOVER | D4D_GRAPH_F_VALUE_X).
- D4D_GRAPH_TXT_PRTY_DEFAULT—This is a text property macro used for default configuration of graph text (value: D4D_TXT_PRTY_ALIGN_H_CENTER_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).

- D4D_GRAPH_FNT_PRTY_DEFAULT — This is a font property macro used for the default configuration of graph font (value: 0).
- D4D_GRAPH_LBL_FNT_PRTY_DEFAULT — This is a font property macro used for the default configuration of graph label font (value: 0).

3.2.9.9.4 D4D_GRAPH Functions

D4D_BOOL D4D_GraphAddTraceData(D4D_OBJECT_PTR pObj, D4D_INDEX trace_ix, D4D_GRAPH_VALUE value);

Input parameters:

- pObj—Pointer to a graph object
- trace_ix—Index of a trace to add data
- value—New value for a selected trace

Output parameters—Result of add operation

Description—The function adds a new sample to one trace, to all the remaining traces it adds the last values.

void D4D_GraphAddTracesData(D4D_OBJECT_PTR pObj, D4D_GRAPH_VALUE* pValues);

Input parameters:

- pObj—Pointer to a graph object
- pValues—Pointer to an input array of new data. The input array has to have the same size as the count of traces of the graph object.

Output parameters—None

Description—The function adds a new sample to all traces of the graph object from input array.

void D4D_GraphClearAll(D4D_OBJECT_PTR pObj);

Input parameters:

- pObj—Pointer to a graph object

Output parameters—NA

Description—Function clears all graph data and graph traces from the screen. The sample index is an erase too.

D4D_GRAPH_SAMPLE_IX D4D_GraphGetSampleIndex(D4D_OBJECT_PTR pObj);

Input parameters:

- pObj—Pointer to a graph object

Output parameters—Index of the last sample of graph traces

Description—Function returns the index of the last sample of graph traces. For example it can be used for generation of labels for the X axis.

D4D_COOR D4D_GraphGetSizeX(D4D_OBJECT_PTR pObj, D4D_SCREEN* pScreen);

Input parameters:

- pObj—Pointer to a graph object
- pScreen—Pointer to the active screen

Output parameters—Dimension of the active area (axis X) of the graph in pixels.

Description—Function returns to the size of the active graph area in pixels of axis X.

D4D_BOOL D4D_GraphSetScaleX(D4D_OBJECT_PTR pObj, Byte mul, Byte div);

Input parameters:

- pObj—Pointer to a graph object
- mul—Multiplier of the scale of axis X
- div—Divider of the scale of axis X

Output parameters—Result of operation, true if it was executed correctly, false if not.

Description—Function sets the new scale of axis X of the graph. It allows to set up a graph object to show more or less samples of the graph active area, than is native to axis X. The scale of axis X is set-up by the ratio of input parameters. To set-up a new scale another function GraphSetDataWidth can be used.

D4D_BOOL D4D_GraphSetDataWidth(D4D_OBJECT_PTR pObj, D4D_INDEX samples);

Input parameters:

- pObj—Pointer to a graph object
- samples—A count of samples shown on the graph

Output parameters—Result of operation, true if it is correctly executed, false if not.

Description—Function sets the new scale of the axis X of the graph. It allows setting up a graph object to show more or less samples of a graph active area, than is native to axis X. The scale of axis X sets-up by the count of the shown samples. To set up a new scale, GraphSetScale can be used too.

void D4D_GraphSetText(D4D_OBJECT_PTR pObj, char* pText);

Input parameters:

- pObj—Pointer to a graph object
- pText—Pointer to a new string

Output parameters—NA

Description—Function changes the title text of a graph. To have success running this function, the original declared text in the instantiation macro has to be placed in RAM.

NOTE

This function is not preferred because version 1.0 and the direct replacement is the general function D4D_SetText with the same parameters and behavior.

3.2.9.9.5 Example of D4D_GRAPH

```
#define GRAPH_LENGTH 200

D4D_GRAPH_VALUE graph_data[GRAPH_LENGTH];
D4D_GRAPH_VALUE graph_data1[GRAPH_LENGTH];
D4D_GRAPH_VALUE graph_data2[GRAPH_LENGTH];
D4D_GRAPH_VALUE graph_data3[GRAPH_LENGTH];

char* Graph_OnNeedLabelText(D4D_OBJECT_PTR pThis, D4D_BOOL axisX, D4D_INDEX gridIx);

D4D_DECLARE_GRAPH_BEGIN(obj_graph, "Graph object example", 5, 5, 300, 200, 3, 3, GRAPH_LENGTH,
D4D_GRAPH_F_DEFAULT, NULL, FONT_8x14, FONT_5x8_SMALL, Graph_OnNeedLabelText, NULL, NULL, NULL)
D4D_DECLARE_GRAPH_TRACE(graph_data, D4D_COLOR_RED, D4D_LINE_THICK, D4D_GRAPH_TRACE_TYPE_LINE)
D4D_DECLARE_GRAPH_TRACE(graph_data1, D4D_COLOR_BLUE, D4D_LINE_THIN, D4D_GRAPH_TRACE_TYPE_DOT)
D4D_DECLARE_GRAPH_TRACE(graph_data2, D4D_COLOR_GREEN, D4D_LINE_THIN,
D4D_GRAPH_TRACE_TYPE_AREA)
D4D_DECLARE_GRAPH_TRACE(graph_data3, D4D_COLOR_ORANGE, D4D_LINE_THICK,
D4D_GRAPH_TRACE_TYPE_AREAINV)
D4D_DECLARE_GRAPH_END()

D4D_DECLARE_SCREEN_BEGIN(screen_graph, ScreenGraph_)
D4D_DECLARE_SCREEN_OBJECT(obj_graph)
D4D_DECLARE_SCREEN_END()



static char* Graph_OnNeedLabelText(D4D_OBJECT_PTR pThis, D4D_BOOL axisX, D4D_INDEX gridIx)
{
    // return the pointer to string that must be drawn on the axis selected by the parameter
axis X an index of //current draw drid line
}
```

3.2.9.10 D4D_SCROLL_BAR

The D4D_SCROOLL_BAR object creates a simple scroll bar that provides standard features:

- Variable position and size
- Horizontal and vertical orientation
- Runtime selectable maximum, minimum, step, and page parameters
- Disable/Enable capability
- Standard keys control
- OnChange callback function to simplify the user application with old and new position
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

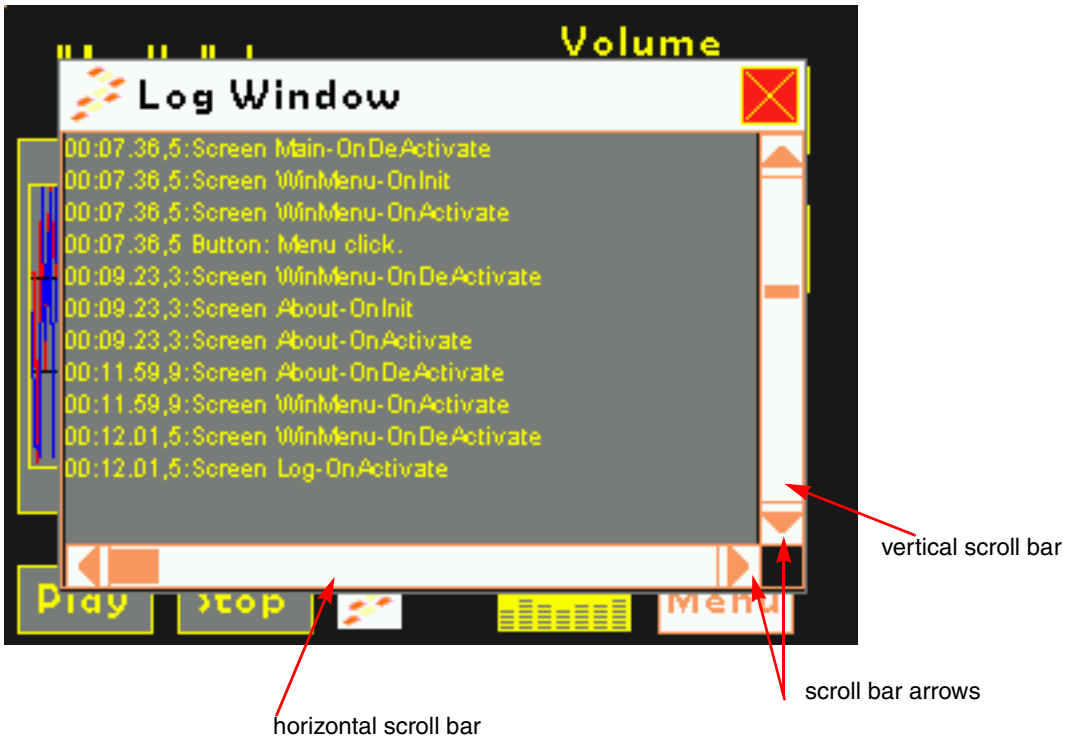


Figure 3-23. Scroll bar objects on the screen example

3.2.9.10.1 D4D_SCROLL_BAR Instantiation Macros

The scroll bar object has a simple instantiation macro in the same shape as most graphic objects that specify all its variable parameters. The driver also contains simpler declaration macros that simplify the full declaration by default object values.

D4D_DECLARE_SCROLL_BAR(name, x, y, cx, cy, flags, pScheme, pUser, pOnChange, pOnUsrMsg)

Input parameters:

- name—Name of a scroll bar object
- x—Position of a scroll bar in the client area of the screen in axis X
- y—Position of a scroll bar in the client area of the screen in axis Y
- cx—Size of a scroll bar in axis X
- cy—Size of a scroll bar in axis Y
- flags—Bitmask that specifies the initial system object flags and scroll bar object flags
- pScheme—Pointer to a color scheme, if its NULL, default scheme is used
- pUser—Pointer on void that can be used by the user application anyway the user wants.
- pOnChange—Pointer to the on change callback function (in format: “void (*OnChange)(D4D_OBJECT* pThis, D4D_INDEX old_position, D4D_INDEX new_position);”). This callback is called when the scroll bar position is changed.

- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped with the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

Description—This macro creates a complete scroll bar object by individually setting all parameters of this object.

D4D_DECLARE_STD_SCROLL_BAR(name, x, y, cx, cy, pOnChange)

Input parameters—All parameters in the declaration have the same pupose as in full declaration macro, the rest of the parameters have default values:

- flags—Has the value of macro D4D_SCRLBR_F_DEFAULT
- pScheme—Has NULL value, the default color scheme will be used
- pUser—Has NULL value.
- pOnUsrMsg—Has NULL value.

Description—This is a default scroll bar definition. The main advantage of this simplified version of declaration macro is less parameters against full base version.

3.2.9.10.2 D4D_SCROLL_BAR Predefined Standard Constants

The scroll bar object contains a few predefined constants that are used in the standard scroll bar declaration. All of these constants can be modified in the user configuration file.

The screen behavior and visual aspect flags:

- D4D_SCRLBR_F_DEFAULT—This is a help macro that is used for default configuration (Value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED | D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FOCUSRECT).
- D4D_SCRLBAR_MIN_TAB_SIZE—This macro specifies the minimum size of the tab of the scroll bar (default value is set to 4 pixels).

3.2.9.10.3 D4D_SCROLL_BAR Functions

void D4D_ScrlBrSetRange(D4D_OBJECT_PTR pObj, D4D_INDEX minimum, D4D_INDEX maximum)

Input parameters:

- pObj—Pointer to a scroll bar object.
- minimum— Minimal value of scroll bar range.
- maximum— Maximal value of scroll bar range.

Output parameters—NA

Description—Sets the range of the scroll bar object. In case that the value of the position is out of the new range, then it is updated to nearest possible value in the new range.

void D4D_ScrlBrSetStep(D4D_OBJECT_PTR pObj, D4D_INDEX page, D4D_INDEX step)

Input parameters:

- pObj—Pointer to a scroll bar object
- page—New value of scroll bar page
- step—New value of scroll bar step

Output parameters—NA

Description—Function sets the step and page parameter of the scroll bar object. The page parameter restricts the whole scroll bar range by page value, but shows on the scroll bar the real page size.

void D4D_ScrlBrSetPosition(D4D_OBJECT_PTR pObj, D4D_INDEX position)

Input parameters:

- pObj—Pointer to a scroll bar object
- position—New value of scroll bar position

Output parameters—NA

Description—Function sets the new position of the scroll bar object. The new position must fit into the scroll bar range. In another situation the position is adapted into a valid range.

D4D_INDEX D4D_ScrlBrGetPosition(D4D_OBJECT_PTR pObj)

Input parameters:

- pObj—Pointer to a scroll bar object

Output parameters:

position—value of the scroll bar position

Description —Returns the position of the scroll bar object

void D4D_ScrlBrChangePosition(D4D_OBJECT_PTR pObj, D4D_INDEX_DELTA change)

Input parameters:

- pObj—Pointer to a scroll bar object
- change—The value of change of the scroll bar position

Output parameters—NA

Description — Function changes the position of the scroll bar object.

3.2.9.10.4 Example of D4D_SCROLL_BAR

```
D4D_DECLARE_STD_SCROLL_BAR_INRAM(scrlBr_Example, 20, 100, 100, 20, D4D_ScrollBarFeedBack)

D4D_DECLARE_SCREEN_BEGIN(screenscrollBar, ScreenScrollBar_)
D4D_DECLARE_SCREEN_OBJECT(scrlBr_Example)
D4D_DECLARE_SCREEN_END()
```

3.2.9.11 D4D_CONSOLE

The D4D_CONSOLE object is prepared to create a simple console that provides standard features:

- Variable position and size
- Horizontal and vertical scroll bars in case that the object size is smaller than the text array
- Cursor function
- Disable/Enable capability
- Standard key controls

pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

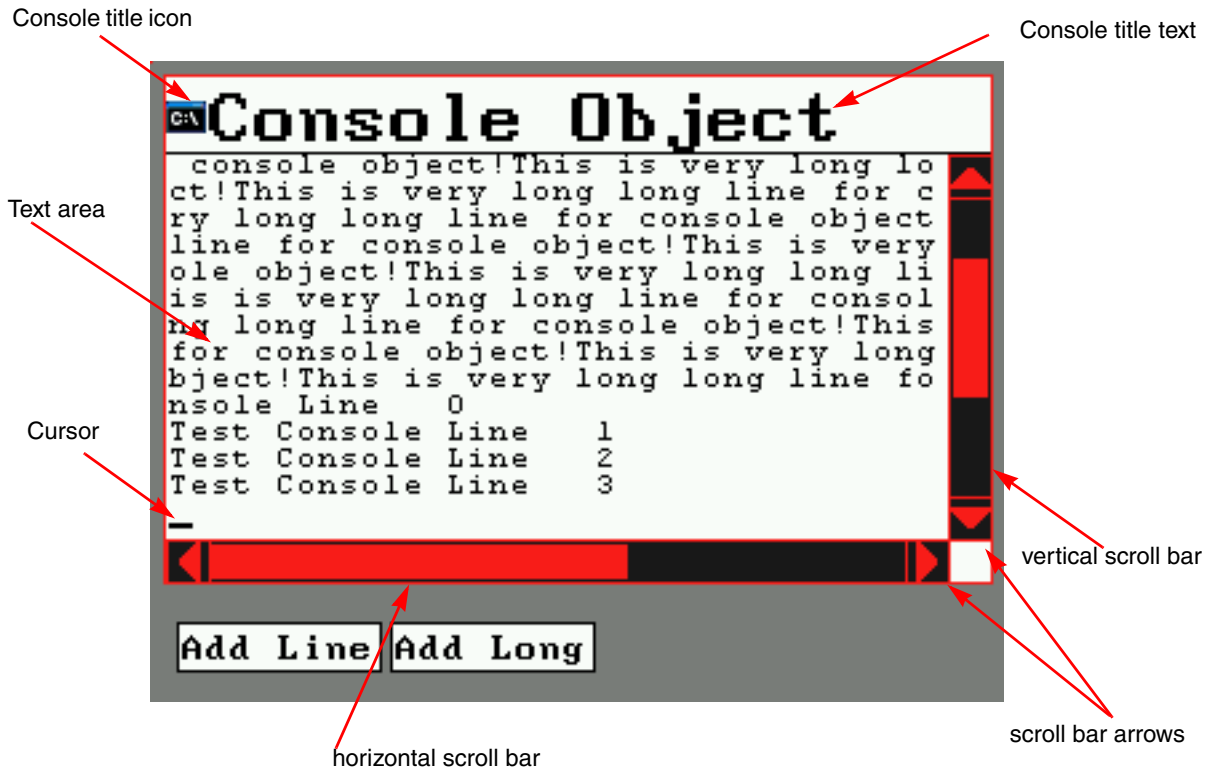


Figure 3-24. Console object on the screen example

3.2.9.11.1 D4D_CONSOLE Instantiation Macros

The console object has a simple instantiation macro in the same shape as most graphic objects that specify all its variable parameters. The driver also contains simpler declaration macros that simplify the full declaration by default object values.

D4D_DECLARE_CONSOLE(name, text, x, y, cx, cy, line_cnt, char_cnt, tabSize, icon, flags, pScheme, fontId, titleFontId, pUser, pOnUsrMsg)

Input parameters:

- name—Name of a console object
- x—Position of a console in the client area of the screen in axis X
- y—Position of a console in the client area of the screen in axis Y
- cx—Size of a console in axis X
- cy—Size of a console in axis Y
- line_cnt—Number of console lines. In case that the text lines take more pixels the console is able to show the vertical scroll bar will be shown.
- char_cnt—Number of console chars per line. In case that chars per line take more pixels than the console is able to show the horizontal scroll bar will be shown.
- tabSize—Count of space chars that are printed instead of the tabulator char
- icon—Title icon
- flags—Bitmask that specifies the initial system object flags and console object flags
- pScheme—Pointer to a color scheme, if it is NULL the default scheme is used
- FontId—Identification number of the used font for the object text
- titleFontId—Identification number of the used font for the object title
- pUser—Pointer on void that can be used by the user application in any manner the user wants
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped with the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

Description—This macro creates a complete console object by individually setting all parameters of this object.

D4D_DECLARE_STD_CONSOLE(name, text, x, y, cx, cy, line_cnt, char_cnt, icon, fontId, titleFontId)

Input parameters—All parameters in the declaration have the same purpose as in a full declaration macro, the rest of the parameters have default values:

- tabSize—Has value of macro D4D_CNSL_STD_TAB_SIZE
- flags—Has value of macro D4D_CNSL_F_DEFAULT
- pScheme—Has NULL value, the default color scheme is used
- pUser—Has NULL value
- pOnUsrMsg—Has NULL value

Description—This is a default console definition. The main advantage of this simplified version of declaration macro is less parameters against full base version.

D4D_DECLARE_STD_CONSOLE_AUTOSIZE(name, text, x, y, line_cnt, char_cnt, icon, fontId, titleFontId)

Input parameters—All parameters in the declaration have the same purpose as in a standard declaration macro, rest parameters (cx, cy) are set to zero to enable the auto size capability.

Description—This is a default console definition with enabled auto size capability. The main advantage is that there are less parameters of the console declaration.

D4D_DECLARE_CONSOLE_AUTOSIZE(name, text, x, y, line_cnt, char_cnt, tabSize, icon, flags, pScheme, fontId, titleFontId, pUser, pOnUsrMsg)

Input parameters—All parameters in the declaration have the same purpose as in a full declaration macro, the rest of the parameters (cx, cy) are set to zero to enable the auto size capability.

Description—This is a console definition with enabled auto size capability. The main advantage of this macro is that it is missing two parameters (sizes).

3.2.9.11.2 D4D_CONSOLE Predefined Standard Constants

The console object contains a few predefined constants that are used in the standard console declaration. All of these constants can be modified in the user configuration file.

The screen behavior and visual aspect flags:

- D4D_CNSL_F_DEFAULT—This is a help macro that is used for default configuration (Value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED | D4D_OBJECT_F_FOCUSRECT | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_TABSTOP).
- D4D_CNSL_TXT_PRTY_DEFAULT—This is a text property macro used for default configuration of the console title font text (Value: D4D_TXT_PRTY_ALIGN_H_LEFT_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).
- D4D_CNSL_FNT_PRTY_DEFAULT—This is a font property macro used for default configuration of the console title font text (Value: 0).
- D4D_CNSL_F_SCRLBRS_DEFAULT—This macro specifies the behavior of console scroll bars, any change is not recommended (Value: D4D_OBJECT_F_ENABLED | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FASTTOUCH | D4D_OBJECT_F_FOCUSRECT).
- D4D_CNSL_SCRLBR_WIDTH—The width of console scroll bars (Default value is 16 pixels)
- D4D_CNSL_SCRLBR_STEP_V—Step of vertical scroll bar (Default value: 2)
- D4D_CNSL_SCRLBR_STEP_H—Step of horizontal scroll bar (Default value: 8)
- D4D_CNSL_CURSOR_HEIGHT—Height of cursor in pixels (Default value is 1 pixel).
- D4D_CNSL_CURSOR_BLINK_TICK_COUNTER—Number of system time ticks to set the blink period of the cursor (Default value is 10 tick). Set the parameter to 0 to hide the cursor.

3.2.9.11.3 D4D_CONSOLE Functions

D4D_BOOL D4D_CnslPutChar(D4D_OBJECT_PTR pObj, D4D_CHAR ch)

Input parameters:

- pObj—Pointer to a console object
- ch— char to draw on cursor

Output parameters—wrap line — D4D_TRUE. The text was wrapped; D4D_FALSE. The text was not wrapped

Description—This function put a new char on the cursor position and increments the cursor position. In case that the cursor achieves the end of line, it is move to a new line and is returned D4D_TRUE value.

D4D_BOOL D4D_CnslPutString(D4D_OBJECT_PTR pObj, D4D_CHAR* pText)

Input parameters:

- pObj—pointer to a console object.
- pText— pointer to text.

Output parameters—wrap line — D4D_TRUE. The text was wrapped; D4D_FALSE. The text was not wrapped

Description—Function puts a new string on the cursor position and updates the cursor position. In case the cursor runs over the end of line, it is returned D4D_TRUE value.

void D4D_CnslClearLine(D4D_OBJECT_PTR pObj, D4D_INDEX line)

Input parameters:

- pObj—Pointer to a console object.
- line— index of line.

Output parameters—NA

Description—Function clears one line selected by the input parameter. Cursor position doe not change.

void D4D_CnslClearAll(D4D_OBJECT_PTR pObj)

Input parameters:

- pObj—Pointer to a console object.

Output parameters—NA

Description—Function clears the complete contents of the console text and sets up the cursor on the first position.

void D4D_CnslGoToXY(D4D_OBJECT_PTR pObj, D4D_POINT newPosition)

Input parameters:

- pObj—Pointer to a console object
- newPosition—New position of cursor

Output parameters—NA

Description—Function sets the new position of the cursor. The point must fit into the range of the text array.

void D4D_CnslEnsureVisible(D4D_OBJECT_PTR pObj)

Input parameters:

- pObj—Pointer to a console object.

Output parameters—NA

Description—Function ensures visibility of the cursor position in the console visible area.

D4D_POINT D4D_CnslGetCursor(D4D_OBJECT_PTR pObj)

Input parameters:

- pObj—Pointer to the console object

Output parameter—Cursor position

Description—Function returns current cursor position.

void D4D_CnslSetScrollBarPosition(D4D_OBJECT_PTR pObj, D4D_INDEX hor, D4D_INDEX ver)

Input parameters:

- pObj—Pointer to the console object
- hor—Position for the horizontal scroll bar
- ver—Position for the vertical scroll bar

Output parameters—NA

Description—Function sets the position of the horizontal and vertical scroll bars.

D4D_POINT D4D_CnslGetScrollBarPosition(D4D_OBJECT_PTR pObj)

Input parameters:

- pObj—Pointer to the console object.

Output parameters—Horizontal and vertical scroll bar position packed in D4D_POINT structure

Description—Function returns the scroll bars position (horizontal and vertical) packed in point structure.

3.2.9.11.4 Example of Use D4D_CONSOLE

```
D4D_DECLARE_STD_CONSOLE(cnslExample, “Console“, 10, 10, 100, 100, 40, 80, NULL, 0, 1)

D4D_DECLARE_SCREEN_BEGIN(screenCnsl, ScreenCNSL_)
D4D_DECLARE_SCREEN_OBJECT(cnslExample)
D4D_DECLARE_SCREEN_END()
```

3.2.9.12 D4D_TEXTBOX

The D4D_TEXTBOX object is prepared to create a simple text box that provides basic features:

- Variable position and size
- Vertical scroll bar in case the object size is smaller than the input text
- Word detection and long word wrapping
- Tabulators support
- New line support
- Disable and enable capability.
- Standard keys control.
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

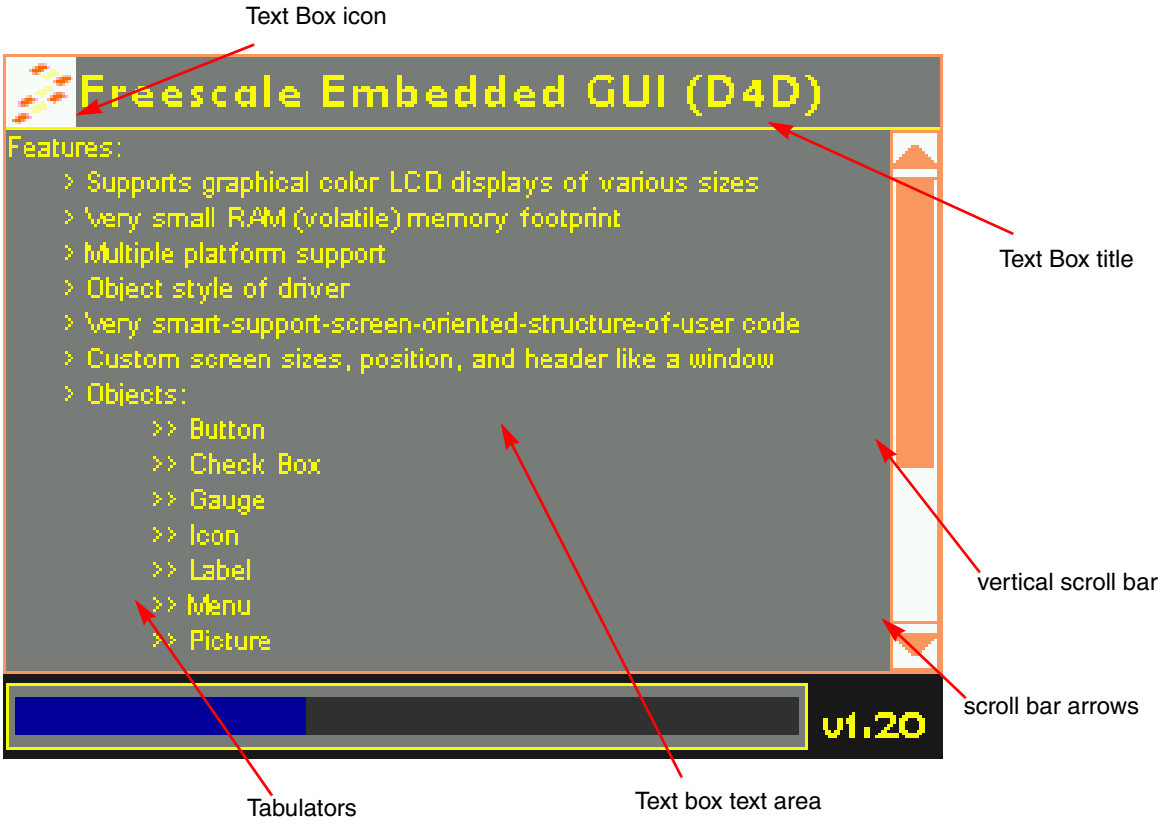


Figure 3-25. Text Box object on the screen example

3.2.9.12.1 D4D_TEXTBOX Instantiation Macros

The text box object has a simple instantiation macro in the same shape as most graphic objects that specify all its variable parameters. The driver also contains simpler declaration macros that simplify the full declaration by the default object values.

D4D_DECLARE_TEXTBOX(name, text, x, y, cx, cy, pTextArray, pTabTable, icon, flags, pScheme, fontId, titleFontId, pUser, pOnUsrMsg)

Input parameters:

- name—Name of a console object
- text—Title text
- x—Position of a text box in the client area of the screen in axis X
- y—Position of a text box in the client area of the screen in axis Y
- cx—Size of a text box in axis X
- cy—Size of a text box in axis Y
- pTextArray—Pointer on text array that must be displayed
- pTabTable—Pointer on the tabulator table
- icon—Title icon
- flags—Bitmask that specifies the initial system object flags and text box object flags
- pScheme—Pointer to a color scheme, if its NULL, default scheme is used.
- FontId—Identification number of the used font for the object text
- titleFontId—Identification number of the used font for the object title
- pUser—Pointer on void that can be used by the user application any way the user wants
- pOnUsrMsg—Pointer to the on-user message callback function (in format: “Byte (*OnUsrMessage)(struct D4D_MESSAGE_S* pMsg);”). This callback is called before this message event is sent to the object itself. The message can be skipped by the D4D_MSG_SKIP return value, in a normal situation the return value must be D4D_MSG_NOSKIP.

Description—This macro creates a complete text box object with individually setting all parameters of this object.

D4D_DECLARE_STD_TEXTBOX(name, text, x, y, cx, cy, pTextArray, pTabTable, icon, fontId, titleFontId)

Input parameters—All parameters in the declaration have the same purpose as in the full declaration macro, the rest of the parameters have default values:

- flags—Has value of macro D4D_TXTBX_F_DEFAULT
- pScheme—Has NULL value, the default color scheme will be used
- pUser—Has NULL value
- pOnUsrMsg—Has NULL value

Description—This is a default text box definition. The main advantage of this simplified version of declaration macro is less parameters against full base version. .

The main advantage of this simplified version of the declaration macro is less parameters against a full base version

D4D_DECLARE_STD_TEXTBOX_AUTOSIZE(name, text, x, y, pTextArray, pTabTable, icon, fontId, titleFontId)

Input parameters—All parameters in the declaration have the same pupose as in a standard declaration macro, the remaining parameters (cx, cy) are set to zero to enable the auto size capability.

Description—This is a default text box definition with enabled auto size capability. The main advantage is there are less parameters of the text box declaration.

D4D_DECLARE_TEXTBOX_AUTOSIZE(name, text, x, y, pTextArray, pTabTable, icon, flags, pScheme, fontId, titleFontId, pUser, pOnUsrMsg)

Input parameters—All parameters in the declaration have the same purpose as in a full declaration macro, the rest of the parameters (cx, cy) are set to zero to enable the auto size capability.

Description—This is a text box definition with enabled auto size capability. The main advantage of this macro is that it is missing two parameters (sizes).

3.2.9.12.2 D4D_TEXTBOX Predefined Standard Constants

The text box object contains a few predefined constants that are used in the standard text box declaration. All of these constants can be modified in the user configuration file.

The screen behavior and visual aspect flags:

- D4D_TXTBX_F_DEFAULT—This is a help macro that is used for default configuration (Value: D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED | D4D_OBJECT_F_FOCUSRECT | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_TABSTOP).
- D4D_TXTBX_TXT_PRTY_DEFAULT—This is a text property macro used for default configuration of the text box title font text (Value: D4D_TXT_PRTY_ALIGN_H_LEFT_MASK | D4D_TXT_PRTY_ALIGN_V_CENTER_MASK).
- D4D_TXTBX_FNT_PRTY_DEFAULT—This is a font property macro used for default configuration of the text box title font text (Value: 0).
- D4D_TXTBX_F_SCRLBRS_DEFAULT—The macro indicates the behavior of the text box scroll bars, any change is not recommended (Value: D4D_OBJECT_F_ENABLED | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FASTTOUCH | D4D_OBJECT_F_FOCUSRECT).
- D4D_TXTBX_SCRLBR_WIDTH—The width of the console scroll bars (Default value is 16 pixels)
- D4D_TXTBX_SCRLBR_STEP—Step of vertical scroll bar (Default value: 2)

3.2.9.12.3 D4D_TEXTBOX Functions

void D4D_TextBoxChangeText(D4D_OBJECT* pObject, D4D_CHAR* pText)

Input parameters:

- pObj—Pointer to a text box object
- pText—Pointer to text

Output parameters—NA

Description—Function changes the source of the text for the text box. When the function is called it changes the source pointer of the text and completely reinicializes the whole object.

void D4D_TextBoxRefreshAll(D4D_OBJECT* pObject)

Input parameters:

- pObj—Pointer to a text box object

Output parameters—NA

Description—Function completely reinicializes the whole object

3.2.9.12.4 Example of Use D4D_TEXTBOX

```
D4D_CHAR pTestText[] = “Test Text Box Text\n \tBullet1\n \t\tBullet2“;

D4D_DECLARE_TAB_TABLE_BEGIN(TabTable)
D4D_DECLARE_TAB(20)
D4D_DECLARE_TAB(50)
D4D_DECLARE_TAB_TABLE_END


D4D_DECLARE_STD_TEXTBOX(txtbxExample, “Text Box“, 10, 10, 100, 100, pTestText, &TabTable, icon,
0, 1)

D4D_DECLARE_SCREEN_BEGIN(screenTxtBx, ScreenTXTBX_)
D4D_DECLARE_SCREEN_OBJECT(txtbxExample)
D4D_DECLARE_SCREEN_END()
```

3.2.10 D4D General Helper Functions

3.2.10.1 Introduction

The D4D also contains a couple of minor helper functions that allow basic drawing, working with text, working with bitmaps, and some other minor work with the driver itself.

3.2.10.2 Basic Drawing Functions

This section contains a couple of basic drawing functions that can be used to add some user-dependent graphics on the screen. For this case D4D_MSG_DRAWDONE can be effectively used.

void D4D_MoveTo(D4D_POINT* ppt);

Input parameters—ppt—Pointer to a point structure type

Output parameters—NA

Description—The function moves the “logical” cursor used for LineTo/LineToXY function by input coordinates. The “logical” cursor is also updated by functions:

- D4D_MoveToXY
- D4D_LineTo / D4D_LineToXY
- D4D_Rect / D4D_RectXY / D4D_RRect / D4D_RRectXY
- D4D_Box / D4D_BoxXY/ D4D_RBox / D4D_RBoxXY
- D4D_FillRect / D4D_FillRectXY/ D4D_FillRRect / D4D_FillRRectXY

void D4D_MoveToXY(D4D_COOR x, D4D_COOR y);

Input parameters:

- x—Coordination in axis X
- y—Coordination in axis Y

Output parameters—NA

Description—The function moves the “logical” cursor that is used for LineTo / LineToXY function by input coordinates. The “logical” cursor is also updated by functions:

- D4D_MoveToXY
- D4D_LineTo / D4D_LineToXY
- D4D_Rect / D4D_RectXY / D4D_RectTo / D4D_RectToXY/D4D_RRect / D4D_RRectXY / D4D_RRectTo / D4D_RRectToXY
- D4D_Box / D4D_BoxXY / D4D_BoxTo / D4D_BoxToXY / D4D_RBox / D4D_RBoxXY / D4D_RBoxTo / D4D_RBoxToXY
- D4D_FillRect / D4D_FillRectXY / D4D_FillRectTo / D4D_FillRectToXY / D4D_FillRRect / D4D_FillRRectXY / D4D_FillRRectTo / D4D_FillRRectToXY

void D4D_LineTo(D4D_POINT* ppt, D4D_LINETYPE ltype, D4D_COLOR color);

Input parameters:

- ppt—Pointer to a point structure type
- ltype—Line type
- color—Color of a line

Output parameters—NA

Description—This function draws a line by the input parameters on the screen. The first point is defined by the last value of the “logical” cursor and the second point of the line is specified by the first parameter of the function. Line type and line color are specified by the next two parameters.

void D4D_LineToXY(D4D_COOR x, D4D_COOR y, D4D_LINETYPE ltype, D4D_COLOR color);

Input parameters:

- x—Coordination in axis X
- y—Coordination in axis Y
- ltype—Line type

- color—Line color

Output parameters—NA

Description—The function draws a line by the input parameters on the screen. The first point is defined by the last value of the “logical” cursor and the second point of line is specified by the first two parameters of the function. The line type and color of the line are specified by the next two parameters.

void D4D_ FillRect(D4D_POINT* ppt, D4D_SIZE* psz, D4D_COLOR color);

void D4D_ FillRRect(D4D_POINT* ppt, D4D_SIZE* psz, D4D_COLOR color, D4D_COOR radius);

Input parameters:

- ppt—Pointer to a point structure type
- psz—Pointer to a size structure size
- color—Color of a line
- radius—Corner radius. Only for the D4D_ FillRRect version of the function

Output parameters—NA

Description—The function draws a filled box that is specified by the top left corner (first parameter—ppt) size of box (second parameter—psz), and color of the box (third parameter—color).

NOTE

The second version of the function draws a filled rectangle with rounded corners.

void D4D_ FillRectXY(D4D_COOR x1, D4D_COOR y1, D4D_COOR x2, D4D_COOR y2, D4D_COLOR color);

void D4D_ FillRRectXY(D4D_COOR x1, D4D_COOR y1, D4D_COOR x2, D4D_COOR y2, D4D_COLOR color, D4D_COOR radius);

Input parameters:

- x1—Coordination of the first point in axis X
- y1—Coordination of the first point in axis Y
- x2—Coordination of the second point in axis X
- y2—Coordination of the second point in axis Y
- color—Color of the line
- radius—Corner radius. Only for the D4D_ FillRRectXY version of function.

Output parameters—NA

Description—The function draws a filled box that is specified by the top left corner (first two parameters—x1 and y1) second point of box (next two parameters—x2 and y2), and the color of a box (last parameter—color).

NOTE

The second version of function draws filled rectangle with rounded corners.

void D4D_FillRectTo(D4D_POINT* ppt, D4D_COLOR color);

void D4D_FillRRectTo(D4D_POINT* ppt, D4D_COLOR color, D4D_COOR radius);

Input parameters:

- ppt—Pointer to a point structure type.
- color—Color of a line
- radius—Corner radius. Only for D4D_ FillRRectTo version of function.

Output parameters—NA

Description—The function draws a filled rectangle that is specified by the top left corner (value of the internal “logical” cursor), the bottom right corner (first parameter—ppt) and fill color (third parameter—color). The function updates the logical cursor by coordination of the bottom right corner.

NOTE

The second version of function draws filled rectangle with rounded corners.

void D4D_FillRectToXY(D4D_COOR x, D4D_COOR y, D4D_COLOR color);

void D4D_FillRRectToXY(D4D_COOR x, D4D_COOR y, D4D_COLOR color, D4D_COOR radius);

Input parameters:

- x—Coordination of the second point in axis X
- y—Coordination of the second point in axis Y
- color—Color of a line
- radius—Corner radius. Only for D4D_ FillRRectToXY version of function.

Description—The function draws a filled rectangle that is specified by the left top corner (value of internal “logical” cursor), bottom right corner (first two parameters x, y) and fill color (third parameter — color). The function updates the logical cursor by coordination of right bottom corner.

NOTE

The second version of function draws filled rectangle with rounded corners.

void D4D_Rect(D4D_POINT* ppt, D4D_SIZE* psz, D4D_LINETYPE ltype, D4D_COLOR color);

void D4D_RRect(D4D_POINT* ppt, D4D_SIZE* psz, D4D_LINETYPE ltype, D4D_COLOR color, D4D_COOR radius);

Input parameters:

- ppt—Pointer to a point structure type
- psz—Pointer to a size structure size
- ltype—Line type
- color—Color of a line

- radius—Corner radius only for the D4D_ RRect version of the function.

Output parameters—NA

Description—The function draws a rectangle that is specified on the top left corner (first parameter—ppt) size of box (second parameter—psz), line type specified by the parameter ltype, and the color of the outline (third parameter—color).

NOTE

The second version of function draws a rectangle with rounded corners.

void D4D_RectXY(D4D_COOR x1, D4D_COOR y1, D4D_COOR x2, D4D_COOR y2, D4D_LINETYPE ltype, D4D_COLOR color);

void D4D_RRectXY(D4D_COOR x1, D4D_COOR y1, D4D_COOR x2, D4D_COOR y2, D4D_LINETYPE ltype, D4D_COLOR color, D4D_COOR radius);

Input parameters:

- x1—Coordination of the first point in axis X
- y1—Coordination of the first point in axis Y
- x2—Coordination of the second point in axis X
- y2—Coordination of the second point in axis Y
- ltype—Line type
- color—Color of a line
- radius—Corner radius. Only for the D4D_ RRectXY version of the function.

Output parameters—NA

Description—The function draws a filled box that is specified by the top left corner (first two parameters—x1 and y1), the bottom right corner (next two parameters—x2 and y2), line type specified by parameter ltype, and color of outline (last parameter—color).

NOTE

The second version of function draws a rectangle with rounded corners.

void D4D_RectTo(D4D_POINT* ppt, D4D_LINETYPE ltype, D4D_COLOR color);

void D4D_RRectTo(D4D_POINT* ppt, D4D_LINETYPE ltype, D4D_COLOR color, D4D_COOR radius);

Input parameters:

- ppt—Pointer to a point structure type
- ltype—Line type
- color—Color of a line
- radius—Corner radius. Only for the D4D_ RRectTo version of the function.

Output parameters—NA

Description—The function draws a rectangle that is specified by the left top corner (value of internal “logical” cursor) size of box (first parameter — ppt), line type specified by the parameter ltype, and color of outline (next parameter — color).

NOTE

The second version of the function draws a rectangle with rounded corners.

void D4D_RectToXY(D4D_COOR x, D4D_COOR y, D4D_LINETYPE ltype, D4D_COLOR color);

void D4D_RRectToXY(D4D_COOR x, D4D_COOR y, D4D_LINETYPE ltype, D4D_COLOR color, D4D_COOR radius);

Input parameters:

- x—Coordination of the second point in axis X
- y—Coordination of the second point in axis Y
- ltype—Line type
- color—Color of a line
- radius—Corner radius. Only for the D4D_ RRectToXY version of the function.

Output parameters—NA

Description—The function draws a filled box that is specified by the left top corner (value of the internal “logical” cursor) right bottom corner (first two parameters—x, y), line type specified by parameter ltype, and color of the outline (last parameter—color).

NOTE

The second version of function draws a rectangle with rounded corners.

void D4D_ Box(D4D_POINT* ppt, D4D_SIZE* psz, D4D_LINETYPE ltype, D4D_COLOR colorLine, D4D_COLOR colorFill);

void D4D_ RBox(D4D_POINT* ppt, D4D_SIZE* psz, D4D_LINETYPE ltype, D4D_COLOR colorLine, D4D_COLOR colorFill, D4D_COOR radius);

Input parameters:

- ppt—Pointer to a point structure type
- psz—Pointer to a size structure size
- ltype—Line type
- colorLine—Color of outline
- colorFill—Fill color
- radius—Corner radius. Only for the D4D_RBox version of the function.

Output parameters—NA

Description—The function draws a filled rectangle that is specified by the top left corner (first parameter—ppt), size of box (second parameter—psz), line type specified by parameter ltype, color of outline (third parameter—colorLine), and color of fill (last parameter—colorFill).

NOTE

The second version of the function draws the box with rounded corners.

void D4D_BoxXY(D4D_COOR x1, D4D_COOR y1, D4D_COOR x2, D4D_COOR y2, D4D_LINETYPE ltype, D4D_COLOR colorLine, D4D_COLOR colorFill);

void D4D_RBoxXY(D4D_COOR x1, D4D_COOR y1, D4D_COOR x2, D4D_COOR y2, D4D_LINETYPE ltype, D4D_COLOR colorLine, D4D_COLOR colorFill, D4D_COOR radius);

Input parameters:

- x1—Coordination of the first point in axis X
- y1—Coordination of the first point in axis Y
- x2—Coordination of the second point in axis X
- y2—Coordination of the second point in axis Y
- ltype—Line type
- colorLine—Color of outline
- colorFill—Fill color
- radius—Corner radius.Only for the D4D_ RBoxXY version of the function.

Output parameters—NA

Description—The function draws a filled rectangle that is specified by the top left corner (first two parameters—x1 and y1) second point of box (next two parameters—x2 and y2), line type specified by parameter ltype, color of outline (third parameter—colorLine), and color of fill (last parameter—colorFill).

NOTE

The second version of the function draws the box with rounded corners.

void D4D_BoxTo(D4D_POINT* ppt, D4D_LINETYPE ltype, D4D_COLOR colorLine, D4D_COLOR colorFill);

void D4D_RBoxTo(D4D_POINT* ppt, D4D_LINETYPE ltype, D4D_COLOR colorLine, D4D_COLOR colorFill, D4D_COOR radius);

Input parameters:

- ppt—Pointer to a point structure type
- ltype—Line type
- colorLine—Color of outline
- colorFill—Fill color
- radius—Corner radius.Only for the D4D_ RBoxTo version of the function.

Output parameters—NA

Description—The function draws a filled box that is specified by the top left corner (value of the internal “logical” cursor), the bottom right corner (first parameter—ppt), line type specified by parameter ltype, color of outline (third parameter—colorLine), and color of fill (last parameter—colorFill).

NOTE

The second version of the function draws a box with rounded corners.

void D4D_BoxToXY(D4D_COOR x, D4D_COOR y, D4D_LINETYPE ltype, D4D_COLOR colorLine, D4D_COLOR colorFill);

void D4D_RBoxToXY(D4D_COOR x, D4D_COOR y, D4D_LINETYPE ltype, D4D_COLOR colorLine, D4D_COLOR colorFill, D4D_COOR radius);

Input parameters:

- x—Coordination of the second point in axis X
- y—Coordination of the second point in axis Y
- ltype—Line type
- colorLine—Color of outline
- colorFill—Fill color
- radius—Corner radius. Only for the D4D_ RBoxToXY version of the function.

Output parameters—NA

Description—The function draws a filled box that is specified by the top left corner (value of the internal “logical” cursor), the bottom right corner (next two parameters—x and y), line type specified by parameter ltype, color of outline (third parameter—colorLine), and color of fill (last parameter—colorFill). The function updates the logical cursor by the bottom right corner coordination.

NOTE

The second version of the function draws the box with rounded corners.

void D4D_FillCircle(D4D_POINT* pCenter, D4D_COOR r, D4D_COLOR color);

Input parameters:

- pCenter—Pointer to a point structure type
- r—Radius
- color—Color of circle

Output parameters—NA

Description—The function draws a circle that is specified by the center point (first parameter—ppt), radius of the circle (second parameter—r), and color (last parameter—color).

void D4D_FillCircleXY(D4D_COOR x, D4D_COOR y, D4D_COOR r, D4D_COLOR color);

Input parameters:

- x—Coordination in axis X
- y—Coordination in axis Y
- r—Radius
- color—Color of circle

Output parameter—NA

Description—The function draws a circle that is specified by the center point (first two parameters—x and y), radius of the circle (second parameter—r), and color (last parameter—color).

void D4D_Circle(D4D_POINT* pCenter, D4D_COOR r, D4D_LINETYPE ltype, D4D_COLOR color);

Input parameters:

- pCenter—Pointer to a point structure type
- r—Radius
- ltype—Line type
- color—Color of circle

Output parameters—NA

Description—The function draws a ring that is specified by the center point (first parameter—ppt), radius of the circle (second parameter—r), line type (third parameter—ltype), and color (last parameter—color).

void D4D_CircleXY(D4D_COOR x, D4D_COOR y, D4D_COOR r, D4D_LINETYPE ltype, D4D_COLOR color);

Input parameters:

- x—Coordination in axis X
- y—Coordination in axis Y
- r—Radius
- color—Color of the circle

Output parameters—NA

Description—The function draws a ring that is indicated by the center point (first two parameters—x and y), radius of the circle (second parameter—r), line type (third parameter—ltype), and color (last parameter—color).

void D4D_FillQuadrant(D4D_POINT* pCenter, D4D_COOR radius, D4D_COLOR color, D4D_QUADRANT quadrant);

Input parameters:

- pCenter—Pointer to a point structure type
- r—Radius
- color—Color of circle
- quadrant—Quadrant index

Output parameters—NA

Description—The function draws one quadrant of the circle that is indicated by the center point (first parameter—ppt), circle radius (second parameter—r), color (last parameter—color), and quadrant index (last parameter quadrant).

void D4D_FillQuadrantXY(D4D_COOR x, D4D_COOR y, D4D_COOR radius, D4D_COLOR color, D4D_QUADRANT quadrant);

Input parameters:

- x—Coordination in axis X
- y—Coordination in axis Y
- r—Radius
- color—Color of circle
- quadrant—Quadrant index

Output parameter—NA

Description—The function draws one quadrant of the circle that is specified by the center point (first two parameters—x, y), circle radius (second parameter—r), color (parameter—color), and quadrant index (last parameter quadrant).

void D4D_Quadrant(D4D_POINT* pCenter, D4D_COOR radius, D4D_LINETYPE ltype, D4D_COLOR color, D4D_QUADRANT quadrant);

Input parameters:

- pCenter—Pointer to a point structure type
- r—Radius
- ltype—Line type
- color—Color of circle
- quadrant—Quadrant index

Output parameters—NA

Description—The function draws a one quadrant of ring that is specified by the center point (first parameter—ppt), radius of circle (second parameter—r), line type (third parameter—ltype), color (parameter—color), and index of the quadrant (last parameter quadrant).

void D4D_QuadrantXY(D4D_COOR x, D4D_COOR y, D4D_COOR radius, D4D_LINETYPE ltype, D4D_COLOR color, D4D_QUADRANT quadrant);

Input parameters:

- x—Coordination in axis X
- y—Coordination in axis Y
- r—Radius
- color—Color of circle

Output parameters—NA

Description—The function draws one quadrant of the ring that is indicated by the center point (first two parameters—x, y), circle radius (second parameter—r), line type (third parameter—ltype), color (parameter—color), and index of the quadrant (last parameter quadrant).

void D4D_DrawBmp(D4D_POINT* ppt, const D4D_BMP* pBmp, D4D_BOOL greyScale);

void D4D_DrawRBmp(D4D_POINT* ppt, const D4D_BMP* pBmp, D4D_BOOL greyScale, D4D_COOR radius);

Input parameters:

- ppt—Pointer to a point structure type
- pBmp—Pointer to a bitmap structure type
- grayScale—Grayscale option
- radius—Corner radius. Only for the D4D_DrawRBmp version of the function.

Output parameters—NA

Description—The function draws a bitmap that is specified by the top left corner (first parameter—ppt), bitmap structure (second parameter—pBmp), and the last option that allows to draw the bitmap in gray scale (last parameter—grayScale).

NOTE

The D4D_DrawRBmp draws the bitmap with rounded corners, but this version does not support all bitmap formats as normal function D4D_DrawBmp. For a list of supported bitmap type, check the configuration file section of bitmaps.

void D4D_DrawBmpXY(D4D_COOR x, D4D_COOR y, const D4D_BMP* pBmp, D4D_BOOL greyScale);

void D4D_DrawRBmpXY(D4D_COOR x, D4D_COOR y, const D4D_BMP* pBmp, D4D_BOOL greyScale, D4D_COOR radius);

Input parameters:

- x—Coordination in axis X
- y—Coordination in axis Y
- pBmp—Pointer to a bitmap structure type
- radius—Corner radius. Only for the D4D_DrawRBmpXY version of the function.

Output parameters—NA

Description—The function draws a bitmap that is specified by the top left corner (first two parameters—x, and y), bitmap structure (second parameter—pBmp), and the last option that allows to draw the bitmap in a gray scale (last parameter grayScale).

NOTE

The D4D_DrawRBmpXY draws a bitmap with rounded corners, but this version does not support all bitmap formats as normal function D4D_DrawBmpXY. For a list of supported bitmap types, check the configuration file section bitmaps.

D4D_SIZE D4D_GetBmpSize(const D4D_BMP* pBmp);

Input parameters:

- pBmp—Pointer to a bitmap

Output parameters—General D4D size structure

Description—The function returns the size of a bitmap that is managed by the input pointer

D4D_COOR D4D_GetBmpWidth(const D4D_BMP* pBmp);

Input parameters:

- pBmp—Pointer to a bitmap

Output parameters—Height of a bitmap

Description—The function returns the height of a bitmap that is managed by the input pointer.

D4D_COOR D4D_GetBmpHeight(const D4D_BMP* pBmp);

Input parameters:

- pBmp—Pointer to a bitmap

Output parameters—Width of a bitmap

Description—The function returns the width of font that is managed by the input pointer.

void D4D_DrawText(D4D_POINT* ppt, D4D_TXTBUFF* buffText, D4D_COLOR colorText, D4D_COLOR colorBkgd);

void D4D_DrawTextTab(D4D_POINT* ppt, D4D_TXTBUFF* buffText, D4D_COOR* pTabTable, D4D_COLOR colorText, D4D_COLOR colorBkgd);

Input parameters:

- ppt—Pointer to a point structure type
- buffText —Pointer to a text buffer will be drawn
- colorText—Color of the text
- colorBkgd—Color of the background
- pTabTable—Pointer on the tabulator table. Used only in the second version of the function.

Output parameters—NA

Description—The function draws a text that is specified by the position of the text (first parameter—ppt), pointer to the text buffer (the text buffer also contains font specification, font, and text properties) (second parameter—buffText), fore, and background color (last two parameters—colorText and colorBkgd).

NOTE

The second version of this function supports printing text with tabulator support.

void D4D_DrawTextXY(D4D_COOR x, D4D_COOR y, D4D_TXTBUFF* buffText, D4D_COLOR colorText, D4D_COLOR colorBkgd);

void D4D_DrawTextTabXY(D4D_COOR x, D4D_COOR y, D4D_TXTBUFF* buffText, D4D_TAB* pTab, D4D_COLOR colorText, D4D_COLOR colorBkgd);

Input parameters:

- x—Coordination in axis X
- y—Coordination in axis Y
- buffText —Pointer to a text buffer will be drawn
- colorText—Color of the text
- colorBkgd—Color of the background
- pTabTable—Pointer on the tabulator table. Used only in the second version of the function.

Output parameters—NA

Description—The function draws a text that is specified by the position of text (first two parameters—x and y), pointer to the text buffer (the text buffer also contains font specification, font, and text properties)(second parameter—buffText), fore, and background color (last two parameters—colorText and colorBkgd).

NOTE

The second version of this function supports printing text with tabulator support.

void D4D_DrawTextRect(D4D_POINT* ppt, D4D_SIZE* psz, D4D_TXTBUFF* buffText, D4D_COLOR colorText, D4D_COLOR colorBkgd);

void D4D_DrawTextRectTab(D4D_POINT* ppt, D4D_SIZE* psz, D4D_TXTBUFF* buffText, D4D_TAB* pTab, D4D_COLOR colorText, D4D_COLOR colorBkgd);

Input parameters:

- ppt—Pointer to a point structure type
- psz—Pointer to a size structure type
- buffText —Pointer to a text buffer will be drawn
- colorText—Color of the text
- colorBkgd—Color of the background
- pTabTable—Pointer on the tabulator table. Used only in the second version of the function.

Output parameters—NA

Description—The function draws a text into a rectangle specified by the position (first parameter—ppt) and size (second parameter—psz), pointer to the text buffer (the text buffer also contains font specification, font, and text properties) (next parameter—buffText), fore, and background color (last two parameters—colorText and colorBkgd).

NOTE

The second version of this function supports printing text with tabulator support.

void D4D_DrawTextRectTo(D4D_POINT* ppt, D4D_TXTBUFF* buffText, D4D_COLOR colorText, D4D_COLOR colorBkgd);

void D4D_DrawTextRectTabTo(D4D_POINT* ppt, D4D_TXTBUFF* buffText, D4D_TAB* pTab, D4D_COLOR colorText, D4D_COLOR colorBkgd);

Input parameters:

- ppt—Pointer to a point structure type
- buffText—Pointer to a text buffer will be drawn
- colorText—Color of the text
- colorBkgd—Color of the background
- pTabTable—Pointer on the tabulator table.Used only in the second version of the function.

Output parameters—NA

Description—The function draws a text into a rectangle specified by the top left corner (the value of the internal “logical” cursor), the bottom right corner (first parameter—ppt), pointer to the text buffer (the text buffer also contains font specification, font, and text properties) (next parameter—buffText), fore, and background color (last two parameters—colorText and colorBkgd).

NOTE

The second version of this function supports printing text with tabulator support.

void D4D_DrawTextRectXY(D4D_COOR x1, D4D_COOR y1, D4D_COOR x2, D4D_COOR y2, D4D_TXTBUFF* buffText, D4D_COLOR colorText, D4D_COLOR colorBkgd);

void D4D_DrawTextRectTabXY(D4D_COOR x1, D4D_COOR y1, D4D_COOR x2, D4D_COOR y2, D4D_TXTBUFF* buffText, D4D_TAB* pTab, D4D_COLOR colorText, D4D_COLOR colorBkgd);

Input parameters:

- x1—Coordination of the top left corner in axis X
- y1—Coordination of the top left corner in axis Y
- x2—Coordination of the bottom right corner in axis X
- y2—Coordination of the bottom right corner in axis Y
- buffText—Pointer to a text buffer will be drawn
- colorText—Color of the text
- colorBkgd—Color of the background
- pTabTable—Pointer on the tabulator table used only in the second version of the function.

Output parameters—NA

Description—The function draws a text into a rectangle specified on the top left corner (first two parameters—x1 and y1), the bottom right corner (next two parameters—x2 and y2), pointer to the text buffer (the text buffer also contains font specification, font, and text properties) (next parameter—buffText), fore, and background color (last two parameters—colorText and colorBkgd).

NOTE

The second version of this function supports printing text with tabulator support.

void D4D_DrawTextRectToXY(D4D_COOR x, D4D_COOR y, D4D_TXTBUFF* buffText, D4D_COLOR colorText, D4D_COLOR colorBkgd);

void D4D_DrawTextRectTabToXY(D4D_COOR x, D4D_COOR y, D4D_TXTBUFF* buffText, D4D_TAB* pTab, D4D_COLOR colorText, D4D_COLOR colorBkgd);

Input parameters:

- x—Coordination of the bottom right corner in axis X
- y—Coordination of the bottom right corner in axis Y
- buffText—Pointer to a text buffer will be drawn
- colorText—Color of the text
- colorBkgd—Color of the background
- pTabTable—Pointer on the tabulator table. Used only in the second version of the function

Output parameters—NA

Description—The function draws a text into rectangle that is specified on the top left corner (value of the internal “logical” cursor), the bottom right corner (first two parameters—x1 and y1), pointer to the text buffer (the text buffer also contains font specification, font, and text properties) (next parameter—buffText), fore, and background color (last two parameters—colorText and colorBkgd)

NOTE.

The second version of this function supports printing text with tabulator support.

3.2.10.3 Strings Helper Functions

This section contains a couple of basic functions that can be used for basic work with strings and fonts.

void D4D_SetText(D4D_OBJECT_PTR pObject, char* pText);

Input parameters:

- pObject—Pointer to an object where text must be changed
- pText—Pointer to a text buffer

Output parameters—NA

Description—The function change text stored in the “main” text buffer of the graphic object. The text buffer is specified by the system internal function specified in objects called GetTextBuffer, in case this function is missing, no text will be changed.

void D4D_SetFontProperties(D4D_OBJECT_PTR pObject, D4D_FONT_PROPERTIES property);

Input parameters:

- pObject—Pointer to an object where text must be changed
- property—New font property

Output parameters—NA

Description—The function change font properties of the text stored in the “main” text buffer of the graphic object. The text buffer is specified by the system internal function specified in objects called GetTextBuffer, in case this function is missing, no text will be changed.

void D4D_SetTextProperties(D4D_OBJECT_PTR pObject, D4D_TEXT_PROPERTIES property);

Input parameters:

- pObject—Pointer to an object where the text must be changed
- property—New text property

Output parameters—NA

Description—The function change text properties of the text stored in the “main” text buffer of the graphic object. The text buffer is specified by the system internal function specified in objects called GetTextBuffer, in case this function is missing, no text will be changed.

Word D4D_GetTextLength(char *pText);

Input parameters:

- pText—Pointer to a text buffer

Output parameters—Length of the input text

D4D_COOR D4D_GetTextBuffWidth(D4D_TXTBUFF* text_buffer);

D4D_COOR D4D_GetTextBuffWidthTab(D4D_TXTBUFF* text_buffer, D4D_TAB* pTab);

Input parameters:

- ix—Font index
- pText—Pointer to a text buffer.
- pTab—Pointer on the tabulator table. Used only in the second version of the function).

Output parameters—Width of the input text in pixels.

Description—The function returns the width of the input text in pixels. The text has to be terminated by zero byte. The differences between this function and D4D_GetTextWidth is that this function measure width of text up to end of text terminated by zero or length stored in D4D_TXTBUFF structure.

NOTE

The second version of this function supports text with tabulator support.

Byte D4D_SprintDecU8(Byte val, char *pText, char fill);

Input parameters:

- val—Value that must be printed to the text
- pText—Pointer to a char buffer where the output text is printed
- fill—This parameter defines char that will be printed before the converted number, if the number is smaller than a full scale function. When at zero, no chars are printed before the number.

Output parameters—Count of printed chars

Description—The function prints the number defined as unsigned char to text

Byte D4D_SprintDecS8(sByte val, char *pText, char fill);

Input parameters:

- val—Value that must be printed to the text
- pText—Pointer to a char buffer where the output text will be printed
- fill—This parameter defines a char that will be printed before the converted number if the number is smaller than a full scale function. When at zero, no chars are printed before the number.

Output parameters—Count of printed chars

Description—The function prints the number defined as signed char to the text.

Byte D4D_SprintDecU16(Word val, char *pi, char fill);

Input parameters:

- val—Value that must be printed to the text
- pText—Pointer to a char buffer where the output text will be printed
- fill—This parameter defines a char that will be printed before the converted number if the number is smaller than a full scale function. When at zero, no chars are printed before the number.

Output parameters—Count of printed chars

Description—The function prints a number defined as unsigned short to text.

Byte D4D_SprintDecS16(sWord val, char *pi, char fill);

Input parameters:

- val—Value that must be printed to the text
- pText—Pointer to a char buffer where the output text will be printed
- fill—This parameter defines a char that will be printed before the converted number if the number is smaller than a full scale function. When at zero, no chars are printed before the number

Output parameters—A count of printed chars

Description—The function prints the number defined as signed short to the text

Byte D4D_SprintDecU32(LWord val, char *pi, char fill);

Input parameters:

- val—Value that must be printed to the text
- pText—Pointer to a char buffer where the output text will be printed
- fill—This parameter defines a char that will be printed before the converted number if the number is smaller than a full scale function. When at zero, no chars are printed before the number.

Output parameters—Count of printed chars

Description—The function prints the number defined as unsigned integer to the text

Byte D4D_SprintDecS32(sLWord val, char *pi, char fill);

Input parameters:

- val—Value that must be printed to the text
- pText—Pointer to a char buffer where the output text will be printed
- fill—This parameter defines a char that will be printed before the converted number if the number is smaller than a full scale function. When at zero, no chars are printed before the number.

Output parameters—Count of printed chars

Description—The function prints the number defined as signed integer to the text

3.2.10.4 General helper functions

This section contains a couple of basic general functions, which can be used for basic work with a driver.

D4D_KEYS D4D_GetKeys(void);

Input parameters—NA

Output parameters—Current state of keys

Description—The function returns current state of input keys

void D4D_ClearKeysBuffer(void);

Input parameters—NA

Output parameters—NA

Description—The function clears all events in an internal input keys buffer

D4D_COLOR D4D_GetCrossColor (D4D_COLOR startColor, D4D_COLOR endColor, Byte value);

Input parameters:

- startColor—Start color
- endColor—End color
- val—Value of the shade of the color

Output parameters—NA

Description—The function computes a shade color from two input colors with a value of shade (0–255). This function can be used for a run-time calculation of the flow change of color.

D4D_COLOR D4D_GetGreyScale (D4D_COLOR color);

Input parameters:

- color—Input color

Output parameters—Converted input color to grayscale

Description—The function converted input color into grayscale

3.2.11 eGUI/D4D Configuration File

The eGUI/D4D contains a configuration file that allows to modify most default values (colors, dimensions, and so on) and change the behavior of the whole driver. The file has to be placed in the user application source file folder and the fastest way how to create it is to use the example that is placed in a subfolder called “Configuration Example” of the driver folder. The file has to be renamed during the copy from “d4d_user_cfg.h.Example” to “d4d_user_cfg.h”).

The configuration file is divided into three main groups:

- General options
- Screen options
- Object options

3.2.11.1 General Options

The general settings part contains the global general settings of basic constants and behavior of the whole D4D. It also contains the low-level driver basic configuration.

3.2.11.1.1 Low-Level Drivers Specification

At first, in the configuration file, low-level drivers types have to be specified. This part of the configuration file is shown below. For the LCD and touch screen drivers the low level drivers are selected in the user application for both layers, see Section Figure 2-1., “Freescale eGUI/D4D block diagram.

```
/*************************************************************//*!
*
* Low Level driver specification for LCD diplay and optionally for
* touch screen
*
*****************************************************************/
// List of implemented low level LCD drivers

// d4dlcd_ssd1289 - driver for LCD displays with SSD1289 controller from Solomon Systech


// Please define a used low LCD driver
#define D4D_LLD_LCD d4dlcd_ssd1289   // the name of low level driver descriptor structure
```

```
// List of implemented low level LCD hw interface drivers

// d4dlcdhw_flexbus_16b - low level hw interface driver for flexbus with 16 bit witdh
// d4dlcdhw_flexbus_8b - low level hw interface driver for flexbus with 8 bit witdh
// d4dlcdhw_gpio6800_8b - low level hw interface driver for gpio drived parallel 8b 6800 bus
// d4dlcdhw_gpio8080_8b - low level hw interface driver for gpio drived parallel 8b 8080 bus
// d4dlcdhw_gpio8080_byte_8b - low level hw interface driver for gpio drived parallel 8b 8080
bus byte version (sends only bytes instead of words as normal)
// d4dlcdhw_spi_8b - low level hw interface driver for hardware SPI with 8 bit witdh
// d4dlcdhw_swspi_16b - low level hw interface driver for software SPI with 16 bit witdh

// Please (if it's needed) define a used LCD hw interface driver
#define D4D_LLD_LCD_HW d4dlcdhw_flexbus_16b   // the name of LCD hw interface driver descriptor
structure


/************************************************************//*!
*
* Touch screen low level driver section
*
*************************************************************/


// List of supported low level touch screen drivers

// d4dtch_resistive - driver for resistive touch screen with direct analog connection to MCU

// Please define a used touch screen driver if touch screen is used in project
#define D4D_LLD_TCH d4dtch_resistive


// List of implemented low level Touch screen hardware interface drivers

// d4dtchhw_s08_adc - low level hardware interface driver for S08 ADC

// Please (if it's needed) define a used touch screen hardware interface driver
#define D4D_LLD_TCH_HW d4dtchhw_s08_adc
```

3.2.11.1.2 Types Definition

The D4D uses a standard name of types as Byte, Word, sWord and so on. If the user project does not contain these sets of types definition, the eGUI/ D4D driver types definition of this standard type definition can be enabled.

```
// The D4D contains own standard types for cases that in the whole project missing
// loaded standard types as Byte, Word, LWord etc.
// To disable using of D4D own standard types change the below define to D4D_TRUE

#define D4D_USE_STANDARD_TYPES D4D_FALSE
```

3.2.11.1.3 Coordination Type Modification

The main D4D_COOR type used in the whole D4D and can be changed with the next option D4D_COOR_TYPE. To use your own type, uncomment the line #define D4D_COOR_TYPE Word and indicate your own (replace Word with another of your own). Default value of D4D_COOR_TYPE is unsigned char.

```
// Variable of coordination’s of display
// in simple – if the display has resolution bigger than 256, than you have to specify
// bigger type than unsigned char (recommended is unsigned short)
// #define D4D_COOR_TYPE Word
```

3.2.11.1.4 Driver Bit Fields Modification

The driver allows modifying internal masks for internal bitfield initializations. With the next two options, the user can adapt the driver for a different bitfield representation in different compilators.

The two options are:

- The align bitfield modification option—D4D_BITFIELD_LSB_ALIGNMENT.
- The size bitfield modification option—D4D_BITFIELD_SHIFT.

```
// Bitfield modification section
// By this options can be modified driver to run with different bitfield representations

     // The first modification is alignment of bitfields sets in compilator right or left align
      #define D4D_BITFIELD_LSB_ALIGNMENT D4D_BITFIELD_LSB_ALLIGMENT_RIGHT
     // The second one allows adapt to 2/4 byte bitfileds for example in Coldfire V2 compilators
      #define D4D_BITFIELD_SHIFT 0

      // Example for Codewarrior Coldfire V2 Compilator
      // #define D4D_BITFIELD_LSB_ALIGNMENT D4D_BITFIELD_LSB_ALLIGMENT_LEFT
      // #define D4D_BITFIELD_SHIFT 24

      // Example for Codewarrior Coldfire V1/S08 Compilator
      // #define D4D_BITFIELD_LSB_ALIGNMENT D4D_BITFIELD_LSB_ALLIGMENT_RIGHT
      // #define D4D_BITFIELD_SHIFT 0
```

3.2.11.1.5 Keys Input Options

The keys input of D4D can be modified with a few parameters in the configuration file. The main options are:

- Length of the input keys buffer. The default value is four—D4D_KEYS_BUFF_LENGTH
- Next options are for modification of the standard and non-standard keys scan codes. All input keys are managed into the driver by the simple keys scan code which has types specified by D4D_KEY_SCANCODE.
- D4D_KEY_FUNC_FOCUS_NEXT—This option allows modification of the key that is dedicated to focus on the next object on the screen. The default key is: D4D_KEY_SCANCODE_RIGHT
- D4D_KEY_FUNC_FOCUS_PREV—This option allows modification of the key that is dedicated to focus the previous object on the screen. The default key is: D4D_KEY_SCANCODE_LEFT.
- Next options are for modification of the standard keys map (all system input keys can be managed into the driver by a simple keys bitmap where the size is specified by D4D_KEYS_TYPE). This option is kept in the driver for only backward compatibility.

```
/*****************************************************************************
*
*   User definition of input KEYS format
*     To apply user definition uncomment the accurate line
*
```

```
*****************************************************************************/

//#define D4D_KEYS_BUFF_LENGTH 8

// Here is place for any change of standard key codes

//#define D4D_KEY_SCANCODE_UP     0x51
//#define D4D_KEY_SCANCODE_DOWN   0x50
//#define D4D_KEY_SCANCODE_LEFT   0x4B
//#define D4D_KEY_SCANCODE_RIGHT  0x4D
//#define D4D_KEY_SCANCODE_ENTER  0x1C
//#define D4D_KEY_SCANCODE_ESC    0x01

//#define D4D_KEY_FUNC_FOCUS_NEXT    D4D_KEY_SCANCODE_RIGHT
//#define D4D_KEY_FUNC_FOCUS_PREV    D4D_KEY_SCANCODE_LEFT

// Backward compatibility section
  //#define D4D_KEY_UP     0x01
  //#define D4D_KEY_DOWN   0x02
  //#define D4D_KEY_LEFT   0x04
  //#define D4D_KEY_RIGHT  0x08
  //#define D4D_KEY_ENTER  0x10
  //#define D4D_KEY_ESC    0x20
```

3.2.11.1.6 Next General Options

The configuration file also contains a few none arrange options that change the behavior of the entire D4D.

List of the remaining general options:

- D4D_MCU_TYPE—This is an important option that sets the MCU platform (default value is ColdFire V1: D4D_MCF51).
- D4D_ENABLE_AUTOSIZE—This defines to turn on or off the auto size capabilities of the D4D driver.
- D4D_ROUND_CORNER_ENABLE—This option enables or disables the round corners of objects and elementary draws.
- D4D_FONT_TABLE_DISABLED—This option enables or disables the font table for the entire driver. When this option is set to true, all strings are disabled.
- D4D_SCREEN_HISTORY—This allows changing the default value of the screen history buffer. The default value is five.
- D4D_SCREEN_SIZE_LONGER_SIDE—This allows changing the default value of the screen size of the longer axis. The default value is 320.
- D4D_SCREEN_SIZE_SHORTER_SIDE—This allows changing the default value of the screen size of the shorter axis. The default value is 240.
- D4D_MCU_BUS_CLOCK—This allows changing the definition of the MCU bus clock that the D4D is using to generate some delays in the low level initialization and calibration of the touch screen

```
/*****************************************************************************
* Constants
*****************************************************************************/
//Select the MCU type that is used in this project supported types:
//D4D_HC08
//D4D_HCS08
//D4D_HC12
//D4D_HCS12
//D4D_HCS12X
//D4D_MCF51
//D4D_MCF52
//D4D_MCF53
//D4D_MCF54
//D4D_MPC51

#define D4D_MCU_TYPE D4D_MCF51

#define D4D_ENABLE_AUTOSIZE D4D_TRUE // This option disables the autosize part of code in a
driver
                                     // It saved in some place in the Flash but all parameters
have to be declared

#define D4D_FONT_TABLE_DISABLED D4D_FALSE   // This option enables and disables the font table
for whole driver                      | | | // When this option is set to true all strings are
disabled

#define D4D_ROUND_CORNER_ENABLE D4D_FALSE  // This option disables and enables round corners
support of the D4D

//#define D4D_SCREEN_HISTORY 10          // This option define depth of history of screens

#define D4D_SCREEN_SIZE_LONGER_SIDE  320  // The maximum resolution of a longer side of physical
LCD

#define D4D_SCREEN_SIZE_SHORTER_SIDE  240  // The maximum resolution of a shorter or longer
side of physical LCD

//#define D4D_MCU_BUS_CLOCK 24000000L    // Mcu bus clock in Hz mainly for few delay loops in
low level

//#define D4D_COLOR_SYSTEM_FORE D4D_COLOR_YELLOW  // System fore color (for example for
calibration screen)

//#define D4D_COLOR_SYSTEM_BCKG D4D_COLOR_BLUE    // System background color (for example for
calibration screen)

//#define D4D_FONT_SYSTEM_DEFAULT 0 // System default font (for example for calibration screen)
```

3.2.11.1.7 eGUI/D4D Callbacks

The D4D has only one callback function that invokes with each input event from the keys and touch screen. The callback can be used for example for any power-saving actions, like managing a backlight of the LCD and other actions.

The callback option is created to enable or disable. The name of this function can be changed by this one option.

The default name of this callback is D4D_InputEventCB.

```
/*************************************************************************
* Callback function definition
*************************************************************************/
#define D4D_INPUT_EVENT_CALLBACK D4D_InputEventCB // The name of the bcallback function
                                                  // that is called with all input events
                                                  // from the keyboard or touch screen.
                                          // Prototype is void D4D_INPUT_EVENT_CALLBACK(void)
                                                  // Uncomment to use it.
```

3.2.11.2 Screen Options

The screen options allow modifying the screen default colors and dimensions.

3.2.11.2.1 Screen Default Colors

The screen has a few default color definitions (to specify the color scheme of the D4D) that need to correct the driver run. These default colors can be changed by these options (uncomment and change value).

- D4D_COLOR_SCR_OUTLINE—Screen outline color
- D4D_COLOR_SCR_TITLEBAR—Screen title bar color
- D4D_COLOR_SCR_TILTLETEXT—Screen title text color
- D4D_COLOR_SCR_EXIT_BTN_FORE—Screen bar exit button forecolor
- D4D_COLOR_SCR_EXIT_BTN_BCKG—Screen bar exit button background color
- D4D_COLOR_SCR_DESKTOP—Screen desktop color

For default values and more details check the D4D screen section.

```
// Define a default screen outline color
//#define D4D_COLOR_SCR_OUTLINE D4D_COLOR_GREY

// Define a default screen header color
//#define D4D_COLOR_SCR_TITLEBAR D4D_COLOR_GREEN

// Define a default screen header text color
//#define D4D_COLOR_SCR_TILTLETEXT D4D_COLOR_WHITE

// Define a default screen header exit button fore color
//#define D4D_COLOR_SCR_EXIT_BTN_FORE D4D_COLOR_YELLOW

// Define a default screen header exit button background color
//#define D4D_COLOR_SCR_EXIT_BTN_BCKG D4D_COLOR_BRIGHT_RED\

// Define a default screen background color
//#define D4D_COLOR_SCR_DESKTOP D4D_COLOR_WHITE
```

3.2.11.2.2 Screen Default Dimensions

The screen has a couple of dimensions that need to correct the function. The default dimensions that are used by the D4D can be changed by setting these options (uncomment and change the value):

D4D_SCR_TITLE_OFF_X—Title text and icon offsets in axis X

D4D_SCR_TITLE_OFF_Y—Title text and icon offsets in axis Y

D4D_SCR_TITLE_EXITBTN_MIN_SIZE—Screen exit button minimum size

D4D_SCR_EXITBTN_CROSS_SIZE— The length of the corner of the screen exit button and the top of the cross inside this button.

D4D_SCR_TITLE_EXITBTN_OFFSET—Screen exit button offset

D4D_SCR_F_DEFAULT—Default setting of the screen used for a standard screen (default value is: 0)

For default values and more details check the D4D screen section.

```
// Define a default title offset in axis X (is used already for title icon offset)
//#define D4D_SCR_TITLE_OFF_X  30

// Define a default title offset in axis Y (is used already for title icon offset)
//#define D4D_SCR_TITLE_OFF_Y  1

// Define a minimum size of EXIT button(cross) in screen header
//#define D4D_SCR_TITLE_EXITBTN_MIN_SIZE  6

// Define a default leasing of cross in EXIT button, opposite to size of button
//#define D4D_SCR_EXITBTN_CROSS_SIZE 2

// Define a default exit button offset
//#define D4D_SCR_TITLE_EXITBTN_OFFSET  4

// Define a default flags
//#define D4D_SCR_F_DEFAULT        (0)
```

3.2.11.2.3 eGUI/D4D Bitmaps Support

The D4D supports various types of bitmaps, but each encoder occupies a space in the flash. The D4D offers options to disable the unused type of bitmaps encoder to save in some place in the flash.

- D4D_BMP_65536NOPAL_ENABLE—65k colors bitmap without a palette
- D4D_BMP_256NOPAL_ENABLE—256 colors bitmap without a palette
- D4D_BMP_PAL_256_ENABLE—256 colors bitmap with a palette
- D4D_BMP_PAL_16_ENABLE—16 colors bitmap with a palette
- D4D_BMP_PAL_2_ENABLE—2 colors bitmap with a palette

```
// 65k colors bitmap without palette
#define D4D_BMP_65536NOPAL_ENABLE D4D_FALSE
// 256 colors bitmap without palette
#define D4D_BMP_256NOPAL_ENABLE D4D_TRUE
// 256 colors bitmap with palette
```

```
#define D4D_BMP_PAL_256_ENABLE D4D_FALSE
// 16 colors bitmap with palette
#define D4D_BMP_PAL_16_ENABLE D4D_FALSE
// 2 colors bitmap with palette
#define D4D_BMP_PAL_2_ENABLE D4D_TRUE
```

3.2.11.3 Objects Options

The configuration file also contains the options for objects. There are two types of these options. The first option is common for all objects and the second option is objects dependent. These options modify the defaults of objects constants.

3.2.11.3.1 Common Object Options

In the common options section there are only general colors of objects. These colors are used for drawing all objects and their general parts.

- D4D_COLOR_FORE_NORM—Standard normal forecolor
- D4D_COLOR_BCKG_NORM—Standard normal background color
- D4D_COLOR_FORE_DISABLED—Standard disabled forecolor
- D4D_COLOR_BCKG_DISABLED—Standard disabled background color
- D4D_COLOR_FORE_FOCUS—Standard focus forecolor
- D4D_COLOR_BCKG_FOCUS—Standard focus background color
- D4D_COLOR_FORE_CAPTURE—Standard capturing forecolor
- D4D_COLOR_BCKG_CAPTURE—Standard capturing background color

```
/*****************************************************************************
* General object colors
*****************************************************************************/

// standard normal color fore
//#define D4D_COLOR_FORE_NORM  D4D_COLOR_BLACK
// standard normal color background
//#define D4D_COLOR_BCKG_NORM  D4D_COLOR_WHITE

// standard disabled color fore
//#define D4D_COLOR_FORE_DISABLED D4D_COLOR_GREY
// standard disabled color background
//#define D4D_COLOR_BCKG_DISABLED D4D_COLOR_LIGHT_GREY

// standard focus color fore
//#define D4D_COLOR_FORE_FOCUS D4D_COLOR_BRIGHT_RED
// standard focus color background
//#define D4D_COLOR_BCKG_FOCUS D4D_COLOR_GREY

// standard capturing color fore
//#define D4D_COLOR_FORE_CAPTURE D4D_COLOR_BRIGHT_GREEN
// standard capturing color background
//#define D4D_COLOR_BCKG_CAPTURE D4D_COLOR_GREEN
```

3.2.11.3.2 Button Options

The button object options:

- Behavior
  - D4D_BTN_F_DEFAULT—This is the default setting of the init flags of the button declaration.
  - D4D_BTN_TXT_PRTY_DEFAULT—This is a default setting of the init text properties of the button declaration. It can be changed in runtime for each object.
  - D4D_BTN_FNT_PRTY_DEFAULT—This is a default setting of init font properties of the button declaration. It can be changed in runtime for each object.
- Appearance
  - D4D_BTN_BORDER_OFFSET—Default text border offset to the border button

```
/*************************************************************************
* Button Object
*************************************************************************/

/**********************
* Properties
**********************/

//#define D4D_BTN_F_DEFAULT (D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED |
D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FOCUSRECT)

//#define D4D_BTN_TXT_PRTY_DEFAULT (D4D_TXT_PRTY_ALIGN_H_CENTER_MASK |
D4D_TXT_PRTY_ALIGN_V_CENTER_MASK)
//#define D4D_BTN_FNT_PRTY_DEFAULT ( 0 )

/**********************
* Sizes constants
**********************/
//#define D4D_BTN_BORDER_OFFSET 3
```

3.2.11.3.3 Gauge Options

The gauge object options:

- Behavior
  - D4D_GAUGE_F_DEFAULT—This is the default setting of the init flags of the gauge declaration.
  - D4D_GAUGE_TXT_PRTY_DEFAULT—This is a default setting of the init text properties of the gauge declaration. It can be changed in runtime for each object.
  - D4D_GAUGE_FNT_PRTY_DEFAULT—This is default setting of init font properties of the gauge declaration. It can be changed in runtime for each object.
- Appearance
  - D4D_GAUGE_HUB_RADIUS—Default radius of gauge hub
  - D4D_COLOR_GAUG_HUB—Default color of gauge hub
  - D4D_COLOR_GAUG_POINTER—Default color of gauge pointer

```
/*************************************************************************
```

```
* Gauge Object
*******************************************************************************/

/**********************
* Properties
**********************/

//#define D4D_GAUGE_F_DEFAULT (D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED |
D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FOCUSRECT |
D4D_GAUGE_F_REDRAW_TEXT | D4D_GAUGE_F_HUB)

//#define D4D_GAUGE_TXT_PRTY_DEFAULT (D4D_TXT_PRTY_ALIGN_H_CENTER_MASK |
D4D_TXT_PRTY_ALIGN_V_CENTER_MASK)
//#define D4D_GAUGE_FNT_PRTY_DEFAULT ( 0 )

/**********************
* Sizes constants
**********************/

//#define D4D_GAUGE_HUB_RADIUS 6

/**********************
* Colors
**********************/
// standard gauge colors
//#define D4D_COLOR_GAUG_HUB        D4D_COLOR_DARK_RED
//#define D4D_COLOR_GAUG_POINTER    D4D_COLOR_DARK_BLUE
```

3.2.11.3.4 Slider Options

The slider object options:

- Behavior
  - D4D_SLDR_F_DEFAULT—This is the default setting of the init flags of the slider declaration.
  - D4D_SLDR_TXT_PRTY_DEFAULT—This is the default setting of the init text properties of the slider declaration. It can be changed in runtime for each object.
  - D4D_SLDR_FNT_PRTY_DEFAULT—This is the default setting of the init font properties of the slider declaration. It can be changed in runtime for each object.
- Appearance
  - D4D_COLOR_SLDR_BAR_FORE—Default forecolor of the slider bar
  - D4D_COLOR_SLDR_BAR_BCKG—Default background color of the slider bar
  - D4D_COLOR_SLDR_BAR_START—Default start forecolor of the slider bar for auto color mode.
  - D4D_COLOR_SLDR_BAR_END—Default end forecolor of slider bar for auto color mode.

```
/*******************************************************************************
* Slider Object
*******************************************************************************/

/**********************
```

```
* Properties
**********************/


//#define D4D_SLDR_F_DEFAULT (D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED |
D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FOCUSRECT |
D4D_OBJECT_F_TRANSP_TEXT)

//#define D4D_SLDR_TXT_PRTY_DEFAULT (D4D_TXT_PRTY_ALIGN_H_CENTER_MASK |
D4D_TXT_PRTY_ALIGN_V_CENTER_MASK)
//#define D4D_SLDR_FNT_PRTY_DEFAULT (D4D_FNT_PRTY_TRANSPARENT_YES_MASK)
/**********************
* Sizes constants
**********************/
//#define D4D_SLDR_BAR_OFF_LENGTH  2
//#define D4D_SLDR_BAR_OFF_WIDTH  4

/**********************
* Colors
**********************/
// standard slider colors
//#define D4D_COLOR_SLDR_BAR_FORE   D4D_COLOR_DARK_BLUE
//#define D4D_COLOR_SLDR_BAR_BCKG   D4D_COLOR_GREY
//#define D4D_COLOR_SLDR_BAR_START  D4D_COLOR_WHITE
//#define D4D_COLOR_SLDR_BAR_END    D4D_COLOR_BLACK
```

3.2.11.3.5 Icon Options

The icon object option:

- Behavior
  - D4D_ICON_F_DEFAULT—This is the default setting of the init flags of the icon declaration.
  - D4D_ICON_TXT_PRTY_DEFAULT—This is the default setting of the init text properties of the icon declaration. It can be changed in runtime for each object.
  - D4D_ICON_FNT_PRTY_DEFAULT—This is the default setting of the init font properties of the icon declaration. It can be changed in runtime for each object.

```
/*****************************************************************************
* Icon Object
*****************************************************************************/

/**********************
* Properties
**********************/

//#define D4D_ICON_F_DEFAULT (D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED)

//#define D4D_ICON_TXT_PRTY_DEFAULT (D4D_TXT_PRTY_ALIGN_H_CENTER_MASK |
D4D_TXT_PRTY_ALIGN_V_CENTER_MASK)
//#define D4D_ICON_FNT_PRTY_DEFAULT ( 0 )
```

3.2.11.3.6 Label Options

The label object option:

- Behavior
  - D4D_LBL_F_DEFAULT—This is the default setting of the init flags of the label declaration
  - D4D_LBL_TXT_PRTY_DEFAULT—This is the default setting of the init text properties of the label declaration. It can be changed in runtime for each object.
  - D4D_LBL_FNT_PRTY_DEFAULT—This is default setting of the init font properties of the label declaration. It can be changed in runtime for each object.

```
/*****************************************************************************
* Label Object
*****************************************************************************/

/***********************
* Properties
***********************/

//#define D4D_LBL_F_DEFAULT (D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED |
D4D_OBJECT_F_FOCUSRECT)

//#define D4D_LBL_TXT_PRTY_DEFAULT (D4D_TXT_PRTY_ALIGN_H_CENTER_MASK |
D4D_TXT_PRTY_ALIGN_V_CENTER_MASK)
//#define D4D_LBL_FNT_PRTY_DEFAULT ( 0 )
```

3.2.11.3.7 Menu Options

The menu object option:

- Behavior
  - D4D_MENU_F_DEFAULT—This is the default setting of the init flags of the menu declaration.
  - D4D_MENU_TXT_PRTY_DEFAULT—This is the default setting of the init text properties of the menu declaration. It can be changed in runtime for each object.
  - D4D_MENU_FNT_PRTY_DEFAULT—This is the default setting of the init font properties of the menu declaration. It can be changed in runtime for each object.
  - D4D_MENU_IX_TXT_PRTY_DEFAULT—This is the setting of the index text properties of the menu declaration. It can not be changed in runtime.
  - D4D_MENU_IX_FNT_PRTY_DEFAULT—This is the setting of the index font properties of the menu declaration. It can not be changed in runtime.
  - D4D_MENU_ITEM_TXT_PRTY_DEFAULT—This is the setting of the items text properties of the menu declaration. It can not be changed in runtime.
  - D4D_MENU_ITEM_FNT_PRTY_DEFAULT—This is the setting of the items font properties of the menu declaration. It can not be changed in runtime.

```
/*****************************************************************************
* Menu Object
*****************************************************************************/
```

```
/**********************
* Properties
**********************/

//#define D4D_MENU_F_DEFAULT (D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED |
D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FOCUSRECT | D4D_MENU_F_INDEX |
D4D_MENU_F_SIDEBAR)

//#define D4D_MENU_TXT_PRTY_DEFAULT (D4D_TXT_PRTY_ALIGN_H_CENTER_MASK |
D4D_TXT_PRTY_ALIGN_V_CENTER_MASK)
//#define D4D_MENU_FNT_PRTY_DEFAULT ( 0 )

//#define D4D_MENU_IX_TXT_PRTY_DEFAULT (D4D_TXT_PRTY_ALIGN_H_CENTER_MASK |
D4D_TXT_PRTY_ALIGN_V_CENTER_MASK)
//#define D4D_MENU_IX_FNT_PRTY_DEFAULT ( 0 )

//#define D4D_MENU_ITEM_TXT_PRTY_DEFAULT D4D_TXT_PRTY_ALIGN_V_CENTER_MASK |
D4D_TXT_PRTY_ALIGN_H_LEFT_MASK)
//#define D4D_MENU_ITEM_FNT_PRTY_DEFAULT ( 0 )
```

3.2.11.3.8 Checkbox Options

The checkbox object options:

- Behavior
  - D4D_CHECKBOX_F_DEFAULT—This is the default setting of the init flags of the checkbox declaration.
  - D4D_MENU_TXT_PRTY_DEFAULT—This is the default setting of the init text properties of the check box declaration. It can be changed in runtime for each object.
  - D4D_MENU_FNT_PRTY_DEFAULT—This is the default setting of init font properties of the check box declaration. It can be changed in runtime for each object.
- Appearance
  - D4D_CHECKBOX_BORDER_OFFSET—Default icon offset to the checkbox border
  - D4D_CHECKBOX_TEXT_OFFSET—Default text offset to the checkbox border
  - D4D_COLOR_CHECKBOX_ICON_BCKG—Default icon background color of the checkbox

```
/*****************************************************************************
* CheckBox Object
*****************************************************************************/

/**********************
* Properties
**********************/

//#define D4D_CHECKBOX_F_DEFAULT (D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED |
D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE\
//                                  | D4D_OBJECT_F_FOCUSRECT | D4D_CHECKBOX_F_ICON_RECTANGLE)

// #define D4D_CHECKBOX_TXT_PRTY_DEFAULT (D4D_TXT_PRTY_ALIGN_H_LEFT_MASK |
D4D_TXT_PRTY_ALIGN_V_CENTER_MASK)
// #define D4D_CHECKBOX_FNT_PRTY_DEFAULT ( 0 )
```

```
/**********************
* Sizes constants
**********************/

//#define D4D_CHECKBOX_BORDER_OFFSET 3
//#define D4D_CHECKBOX_TEXT_OFFSET 4


/**********************
* Colors
**********************/
// standard check box colors
//#define D4D_COLOR_CHECKBOX_ICON_BCKG D4D_COLOR_WHITE
```

3.2.11.3.9 Graph Options

The graph object option:

- Behavior
  - D4D_GRAPH_F_DEFAULT—This is the default setting of init flags of the graph declaration
  - D4D_GRAPH_TXT_PRTY_DEFAULT—This is the default setting of the init text properties of the graph declaration. It can be changed in runtime for each object.
  - D4D_GRAPH_FNT_PRTY_DEFAULT—This is the default setting of the init font properties of the graphic declaration. It can be changed in runtime for each object.
  - D4D_GRAPH_LBL_FNT_PRTY_DEFAULT—This is the setting of the labels font properties of the menu declaration. It can not be changed in runtime.
- Appearance
  - D4D_GRAPH_BORDER_OFF—Default graph offset between active area and the graph object border
  - D4D_GRAPH_VALUE_OFF—Default graph labels offsets in a graph object
  - D4D_COLOR_GRAPH_GRID—Default graph grid color

```
/*****************************************************************************
* Graph Object
*****************************************************************************/

/**********************
* Properties
**********************/

//#define D4D_GRAPH_F_DEFAULT (D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED |
D4D_OBJECT_F_FOCUSRECT | D4D_GRAPH_F_MODE_ROLLOVER | D4D_GRAPH_F_VALUE_X_BOTT |
D4D_GRAPH_F_VALUE_Y_RIGHT)

//#define D4D_GRAPH_TXT_PRTY_DEFAULT (D4D_TXT_PRTY_ALIGN_H_CENTER_MASK |
D4D_TXT_PRTY_ALIGN_V_CENTER_MASK)
//#define D4D_GRAPH_FNT_PRTY_DEFAULT ( 0 )
//#define D4D_GRAPH_LBL_FNT_PRTY_DEFAULT ( 0 )

/**********************
* Sizes constants
**********************/
```

```
//#define D4D_GRAPH_BORDER_OFF        5
//#define D4D_GRAPH_VALUE_OFF        2


/***********************
* Colors
***********************/
// standard graph grid color
//#define D4D_COLOR_GRAPH_GRID D4D_COLOR_GREY
```

3.2.11.3.10 Scroll Bar Option

The graph object option:

- Behavior
  - D4D_SCRLBR_F_DEFAULT—This is the default setting of init flags of the scroll bar declaration.
- Appearance
  - D4D_SCRLBAR_MIN_TAB_SIZE—Minimal size of the scroll bar tab in pixels.

```
/*****************************************************************************
* Scroll bar Object
*****************************************************************************/

/***********************
* Properties
***********************/

//#define D4D_SCRLBR_F_DEFAULT (D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED |
D4D_OBJECT_F_TABSTOP | D4D_OBJECT_F_TOUCHENABLE | D4D_OBJECT_F_FOCUSRECT)

/***********************
* Sizes constants
***********************/
//#define #define D4D_SCRLBAR_MIN_TAB_SIZE ( 4 )
```

3.2.11.3.11 Text Box Option

The graph object option:

- Behavior
  - D4D_TXTBX_F_DEFAULT—This is the default setting of the init flags of the text box declaration.
  - D4D_TXTBX_F_SCRLBRS_DEFAULT—This is the default setting of init flags of the text box scroll bars declaration.
  - D4D_TXTBX_TXT_PRTY_DEFAULT—This is the default setting of the init text properties of the text box declaration. It can be changed runtime for each object.
  - D4D_TXTBX_FNT_PRTY_DEFAULT—This is the default setting of the init font properties of the text box declaration. Runtime for each object can be changed.

- Appearance
  - D4D_TXTBX_SCRLBR_WIDTH—Width of the text box scroll bars in pixels
  - D4D_TXTBX_SCRLBR_STEP_V—Text box scroll bar step for vertical/line movement

```
/*****************************************************************************
* Console Object
*****************************************************************************/

/***********************
* Properties
***********************/
//Console default properties
//#define D4D_CNSL_F_DEFAULT (D4D_OBJECT_F_VISIBLE | D4D_OBJECT_F_ENABLED |
D4D_OBJECT_F_FOCUSRECT | D4D_OBJECT_F_TOUCHENABLE)

// Console scroll bars default properties
//#define D4D_CNSL_F_SCRLBRS_DEFAULT (D4D_OBJECT_F_ENABLED | D4D_OBJECT_F_TOUCHENABLE |
D4D_OBJECT_F_FASTTOUCH | D4D_OBJECT_F_FOCUSRECT)

//#define D4D_CNSL_TXT_PRTY_DEFAULT (D4D_TXT_PRTY_ALIGN_H_LEFT_MASK |
D4D_TXT_PRTY_ALIGN_V_CENTER_MASK)
//#define D4D_CNSL_FNT_PRTY_DEFAULT ( 0 )

/***********************
* Sizes constants
***********************/
//#define D4D_CNSL_TAB_SIZE  ( 8 )
//#define D4D_CNSL_SCRLBR_WIDTH  ( 16 )
//#define D4D_CNSL_SCRLBR_STEP_V  ( 2 )
//#define D4D_CNSL_SCRLBR_STEP_H  ( 8 )
```

3.3 Low-Level Drivers To eGUI/D4D API

3.3.1 Introduction

The D4D is a high-level graphic driver that provides end user function and does not have direct access to the LCD interface. It needs a low-level driver that provides an interface and basic functions to control the LCD. The low-level driver also provides some basic types. The eGUI/D4D supports both types of low-level drivers:

- LCD controllers are mandatory
- Touch screen driver is optional. The D4D can run without a touch screen driver.

3.3.2 Low-Level LCD/Touch Screen Drivers

The D4D library can support lots of LCD types with a system of low-level drivers. The supported LCD types have to have their own RAM memory and provide the MCU data interface. The structure of D4D low level drivers are designed to be as flexible as possible to allow to connect different LCDs and their interfaces. The D4D can be also implemented in any RTOS like the MQX and uses low level interface.

The low-level driver has to provide only software API for interaction with the D4D.

3.3.2.1 D4D Low-Level API

The D4D low level drivers (both group LCD and touch screen) are divided into two layers (as shown in Figure 3-26) that manage control and communication with the LCD and touch screen. The low level drivers are divided for better flexibility with porting to another LCD and touch screen or hardware to minimize the developing time. For all the low level drivers there are also prepared template files for speeding up and creating a new driver.

LCD low level drivers layers:

- LCD controller layer—This is a layer that directly manages all D4D driver needs with cooperation from the LCD. This layer is mandatory and must be used in the project. This layer solves all the the things dependent on the LCD, it can also be an implemented hardware communication part of the work. Implementing communication part of the LCD control is not recommended into this layer due to a better porting option. This driver must contain a filled structure. D4DLCD_FUNCTIONS (defined in d4d_lldapi.h) which is used as an API to D4D driver.
- Hardware communication layer—This layer provides interface between the LCD controller layer and the MCU hardware. This is an optional driver that can be a walk around direct connection from the higher layer to the hardware. This option is not recommended. This driver must contain a filled structure D4DLCDHW_FUNCTIONS (defined in d4d_lldapi.h) which is used as an API to the LCD controller layer.

Touch screen low level driver layers:

- Touch screen control layer—This layer directly manages all D4D driver needs for touch screen cooperation,. In case that the Touch screen is used in the project this layer is mandatory and it must be used in the project. This layer solves all the things dependent on the touch screen control. It can also contain the hardware interface layer. Implementing the hardware connection part of the touch screen control is not recommended into this layer due to a better porting option. This driver must contain filled structure D4DTCH_FUNCTIONS (defined in d4d_lldapi.h) which is used as an API to D4D driver.
- Hardware communication layer—This layer provides an interface between the touch screen control layer and the MCU hardware. This is an optional driver that can be a walk around direct connection to the higher layer hardware. This option is not recommended. This driver must contain filled structure D4DTCHHW_FUNCTIONS (defined in d4d_lldapi.h) which is used as an API to the touch screen control layer.

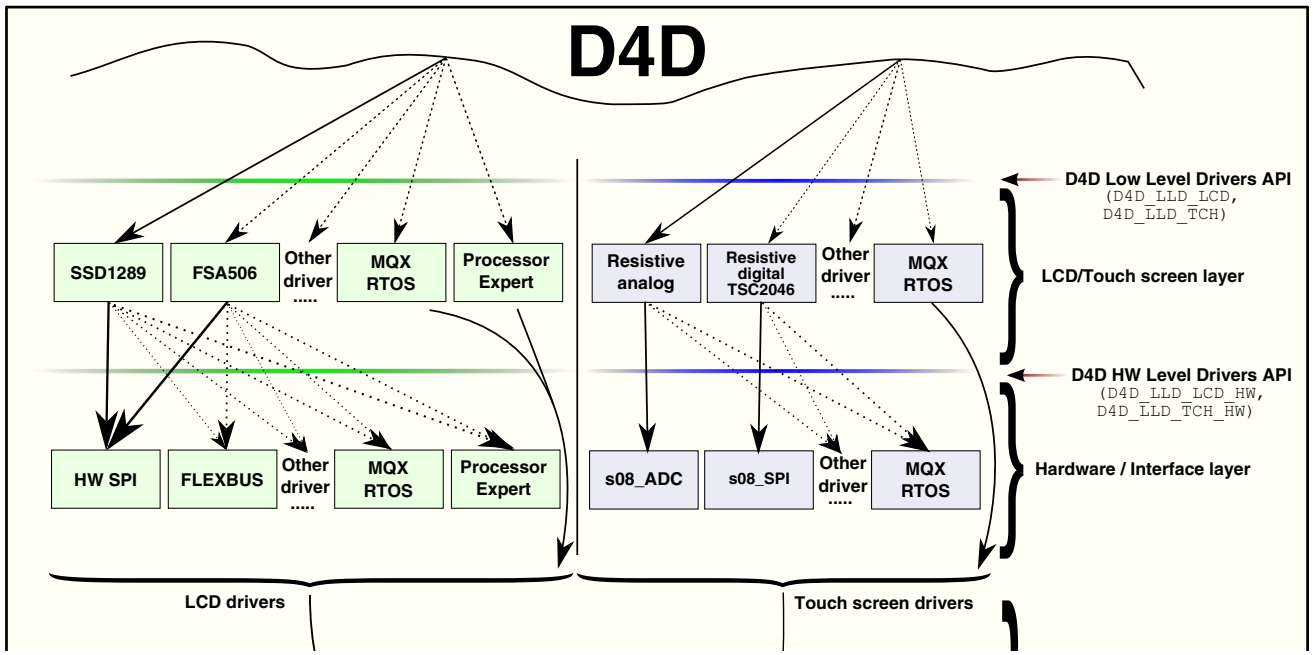


Figure 3-26. D4D Low Level drivers structure

3.3.2.2 eGUI/D4D Low-Level API Types

The eGUI/D4D library provides all low level API types in the dedicated header file d4d_lldapi.h.

Table 3-2. Name

Type	Description

D4DLCD_ORIENTATION	This type is used for orientation of the LCD display. It is defined as enum and synchronized with D4D_ORIENTATION type used in entire D4D library.
	`typedef enum`
	`{`
	`Portrait = D4D_ORIENT_PORTRAIT,`
	`Portrait180 = D4D_ORIENT_PORTRAIT180,`
	`Landscape = D4D_ORIENT_LANDSCAPE,`
	`Landscape180 = D4D_ORIENT_LANDSCAPE180`
	`} D4DLCD_ORIENTATION;`

D4DHW_PIN_STATE	This is enum type used for managing actions with the pin to pin control function defined in Low Level API.
	`typedef enum`
	`{`
	`D4DHW_PIN_PULL_UP_ON,`
	`D4DHW_PIN_PULL_UP_OFF,`
	`D4DHW_PIN_OUT,`
	`D4DHW_PIN_IN,`
	`D4DHW_PIN_SET_0,`
	`D4DHW_PIN_SET_1,`
	`D4DHW_PIN_GET,`
	`D4DHW_PIN_ADC_ON,`
	`D4DHW_PIN_ADC_OFF`
	`}D4DHW_PIN_STATE`

D4DLCDHW_PINS	This is enum type that contains the LCD control pins that manage the pin name to low level functions.
	`typedef enum`
	`{`
	`D4DLCD_RESET_PIN,`
	`D4DLCD_BACKLIGHT_PIN`
	`}D4DLCDHW_PINS;`

D4DTCHHW_PINS	This is enum type that contains touch screen control pins that manages the pin name to low level functions.
	`typedef enum`
	`{`
	`D4DTCH_X_PLUS_PIN,`
	`D4DTCH_X_MINUS_PIN,`
	`D4DTCH_Y_PLUS_PIN,`
	`D4DTCH_Y_MINUS_PIN`
	`}D4DTCHHW_PINS;`

Type	Description

D4DLCD_FUNCTIONS	This is an API structure that contains all the necessary pointers to the functions used as API between the D4D library and the LCD controller layer. This filled structure has to contain each LCD controller low level driver compatible with the D4D library.
	`typedef struct D4DLCD_FUNCTIONS_S`
	`{`
	`  unsigned char (*D4DLCD_Init)(void);`
	`  unsigned char (*D4DLCD_SetWindow)(unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2);`
	`  unsigned char (*D4DLCD_SetOrientation)(D4DLCD_ORIENTATION new_orientation);`
	`  unsigned char (*D4DLCD_Send_PixelColor)(unsigned short value);`
	`  unsigned short (*D4DLCD_Read_Word)(void);`
	`  void (*D4DLCD_FlushBuffer)(void);`
	`  void (*D4DLCD_Delay_ms)(unsigned short period);`
	`  unsigned char (*D4DLCD_DeInit)(void);`
	`}D4DLCD_FUNCTIONS;`

D4DLCDHW_FUNCTIONS	This is an API structure that contains all the necessary pointers to the functions that are used as API between LCD controller layer and LCD hardware communication layer.
	`typedef struct D4DLCDHW_FUNCTIONS_S`
	`{`
	`  unsigned char (*D4DLCDHW_Init)(void);`
	`  unsigned char (*D4DLCDHW_SendDataWord)(unsigned short Value);`
	`  unsigned char (*D4DLCDHW_SendCmdWord)(unsigned short Cmd);`
	`  unsigned short (*D4DLCDHW_ReadDataWord)(void);`
	`  unsigned short (*D4DLCDHW_ReadCmdWord)(void);`
	`  unsigned char (*D4DLCDHW_PinCtl)(D4DLCDHW_PINS pinId, D4DHW_PIN_STATE setState);`
	`  void (*D4DLCDHW_FlushBuffer)(void);`
	`  unsigned char (*D4DLCDHW_DeInit)(void);`
	`}D4DLCDHW_FUNCTIONS;`

D4DLCDHWFB_FUNCTIONS	This is an API structure that contains all neccessy pointers to functions that are used as API between the frame buffer LCD controller layer and the LCD frame buffer peripheral communication layer.
	`typedef struct D4DLCDHWFB_FUNCTIONS_S`
	`{`
	`  unsigned char (*D4DLCDHW_Init)(void);`
	`  unsigned char (*D4DLCDHW_WriteData)(unsigned long addr, unsigned short value);`
	`  unsigned short (*D4DLCDHW_ReadData)(unsigned long addr);`
	`  D4DLCD_FRAMEBUFF_DESC* (*D4DLCDHW_GetFbDescriptor)(void);`
	`  unsigned char (*D4DLCDHW_PinCtl)(D4DLCDHW_PINS pinId, D4DHW_PIN_STATE setState);`
	`  void (*D4DLCDHW_FlushBuffer)(void);`
	`  unsigned char (*D4DLCDHW_DeInit)(void);`
	`}D4DLCDHWFB_FUNCTIONS;`

Type	Description

D4DLCD_FRAMEBUFF_DESC	This structure describes all the necessary properties of the frame buffer display, mainly the pointer to the frame buffer in the memory, resolution, and color depth.
	`typedef struct D4DLCD_FRAMEBUFF_DESC_S`
	`{`
	`  unsigned long fb_start_addr;`
	`  unsigned short lcd_x_max;`
	`  unsigned short lcd_y_max;`
	`  unsigned char bpp_byte;`
	`}D4DLCD_FRAMEBUFF_DESC;`

D4DTCH_FUNCTIONS	This is an API structure that contains all the necessary pointers to functions that are used as API between the D4D library and touch screen control layer. This filled structure has to contain each touch screen control low level driver compatible with the D4D library.
	`typedef struct D4DTCH_FUNCTIONS_S`
	`{`
	`  unsigned char (*D4DTCH_Init)(void);`
	`  unsigned char (*D4DTCH_GetPositionRaw)(unsigned short *TouchPositionX, unsigned short *TouchPositionY);`
	`  D4D_TOUCHSCREEN_LIMITS* (*D4DTCH_GetRawLimits)(void);`
	`  unsigned char (*D4DTCH_DeInit)(void);`
	`}D4DTCH_FUNCTIONS;`

D4DTCHHW_FUNCTIONS	This is an API structure that contains all the necessary pointers to the functions that are used as API between the touch screen control layer and the touch screen hardware control layer.
	`typedef struct D4DTCHHW_FUNCTIONS_S`
	`{`
	`  unsigned char (*D4DTCHHW_Init)(void);`
	`  unsigned short (*D4DTCHHW_ReadTouchAxis)(D4DTCHHW_PINS pinId);`
	`  D4D_TOUCHSCREEN_LIMITS* (*D4DTCHHW_GetRawLimits)(void);`
	`  unsigned char (*D4DTCHHW_PinCtl)(D4DTCHHW_PINS pinId, D4DHW_PIN_STATE setState);`
	`  unsigned char (*D4DTCHHW_DeInit)(void);`
	`}D4DTCHHW_FUNCTIONS;`

D4D_TOUCHSCREEN_LIMITS	This is the structure used to store the touch screen hardware dependent hardware to properly run the function of the calibration touch screen and recalculating raw touch values into pixel coordination.
	`typedef struct D4D_TOUCHSCREEN_LIMITS_S`
	`{`
	`  unsigned short rawDataScale;`
	`  unsigned short touchMinX;`
	`  unsigned short touchMinY;`
	`  unsigned short touchOffMaxX;`
	`  unsigned short touchOffMaxY;`
	`}D4D_TOUCHSCREEN_LIMITS;`

3.3.2.3 Low-Level LCD API Functions Prototypes

This part describes all the functions the prototypes need to run the LCD low level drivers interfaces.

3.3.2.3.1 LCD Controller Layer API

unsigned char (*D4DLCD_Init)(void);

Input parameters—NA

Output parameters—Results of the initialization process. 0 for false, 1 for success.

Description—The function initializes all the necessary things to run the LCD hardware and low-level driver software.

unsigned char (*D4DLCD_SetWindow)(unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2);

Input parameters:

- x1—First point coordination in axis X
- y1—First point coordination in axis Y
- x2—First point coordination in axis X
- y2—First point coordination in axis Y

Output parameters—Result of the set window operation. 0 for false, 1 for success.

Description—The function creates a logic window in the entire physical screen and prepares the LCD controller for receiving individual pixels into this window.

unsigned char (*D4DLCD_SetOrientation)(D4DLCD_ORIENTATION new_orientation);

Input parameters:

- new_orientation—New orientation of the LCD

Output parameters—Result of the set orientation operation. 0 for false, 1 for success.

Description—The function changes the orientation of the LCD driver. The orientation can assume all the states that contain the D4DLCD_ORIENTATION enumeration type defined in d4d_lldapi.h file

unsigned char (*D4DLCD_Send_PixelColor)(unsigned short value);

Input parameters:

- value—Color of pixel

Output parameters—Result of the send pixel operation. 0 for false, 1 for success.

Description—The function sends one color pixel to display. The pixel is written into a logic window created by the function D4DLCD_SetWindow.

unsigned short (*D4DLCD_Read_Word)(void);

Input parameters—NA

Output parameters—Read word value

Description—The function reads the word from the selected logic window by function D4DLCD_SetWindow on LCD.

void (*D4DLCD_FlushBuffer)(void);

Input parameters—NA

Output parameters—NA

Description—The function is used with buffered hardware interfaces. The D4D library calls this function each time. When it is finished draw each basic graphic item.

void (*D4DLCD_Delay_ms)(unsigned short period);

Input parameters:

- period—Count of ms to wait

Output parameters—NA

Description—The function must wait (stop the MCU) for time specified in the parameter. This time delaying the D4D library is used only during initialization and touchscreen calibration, so the D4D does not do any senseless delay in a normal run.

unsigned char (*D4DLCD_DeInit)(void);

Input parameters—NA

Output parameters—Result of deinitialization process. 0 for false, 1 for success.

Description—The function must deinitialize all things initialized by the LCD hardware and low-level driver software.

3.3.2.3.2 LCD Hardware Communication Layer API

This part contains function prototypes that are used to communication between the LCD controller layer and hardware communication layer.

unsigned char (*D4DLCDHW_Init)(void);

Input parameters—NA

Output parameters—Result of the initialization process. 0 for false, 1 for success.

Description—The function initializes all the necessary things to run the LCD communication hardware.

unsigned char (*D4DLCDHW_SendDataWord)(unsigned short value);

Input parameters:

- value—16-bit data.

Output parameters—Result of the send word operation. 0 for false, 1 for success.

Description—The function sends a 16-bit data word to display.

unsigned char (*D4DLCDHW_SendCmdWord)(unsigned short cmd);

Input parameters:

- cmd—16-bit command.

Output parameters—Result of the send command word operation. 0 for false, 1 for success.

Description—The function sends 16-bit command word to display.

unsigned short (*D4DLCDHW_ReadDataWord)(void);

Input parameters—NA

Output parameters—Read 16-bit data word from LCD

Description—The function reads one data word from LCD.

unsigned short (*D4DLCDHW_ReadCmdWord)(void);

Input parameters—NA

Output parameters—Read 16-bit command word from LCD

Description—The function reads one command word from LCD.

unsigned char (*D4DLCDHW_PinCtl)(D4DLCDHW_PINS pinId, D4DHW_PIN_STATE setState);

Input parameters:

- pinId—Pin identification
- setState—Action to do with selected pin

Output parameters—For read pin operation return pin value

Description—The function controls pins that are not directly in association with communication interface.

void (*D4DLCDHW_FlushBuffer)(void);

Input parameters—NA

Output parameters—NA

Description—The function is to be used with buffered hardware interfaces. In the LCD control layer, call this function in case the last write word operation was last for the current write sequence.

unsigned char (*D4DLCDHW_DeInit)(void);

Input parameters—NA

Output parameters—Result of deinitialization process. 0 for false, 1 for success.

Description—The function deinitializes the LCD communication hardware driver.

3.3.2.3.3 LCD frame buffer peripheral communication layer API

This part contains function prototypes that are used to communication between the LCD controller layer and frame buffer peripheral communication layer.

unsigned char (*D4DLCDHW_Init)(void);

Input parameters—NA

Output parameters—Result of the initialization process. 0 for false, 1 for success.

Description—The function initializes all the necessary things to run the LCD frame buffer peripheral.

unsigned char (*D4DLCDHW_WriteData)(unsigned long addr, unsigned short value);

Input parameters:

- addr—Address of pixel in memory
- value—16-bit data.

Output parameters—Result of write word operation. 0 for false, 1 for success.

Description—The function writes a 16-bit data word to the MPU memory (frame buffer).

unsigned short (*D4DLCDHW_ReadData)(unsigned long addr);

Input parameters:

- addr—Address of the pixel in the memory

Output parameters—Reads 16-bit data word from the MPU memory (frame buffer)

Description—The function reads the one data word from the MPU memory (frame buffer)

D4DLCD_FRAMEBUFF_DESC* (*D4DLCDHW_GetFbDescriptor)(void);

Input parameters—NA

Output parameters—Pointer on the frame buffer descriptor structure (D4DLCD_FRAMEBUFF_DESC)

Description—The function return pointer on the frame buffer descriptor structure (D4DLCD_FRAMEBUFF_DESC) that contains all necessary data about frame buffer display properties.

unsigned char (*D4DLCDHW_PinCtl)(D4DLCDHW_PINS pinId, D4DHW_PIN_STATE setState);

Input parameters:

- pinId—Pin identification
- setState—Action to do with the selected pin

Output parameters—For read, pin operation return pin value

Description—The function is to control pins that are not directly in association with the communication interface.

void (*D4DLCDHW_FlushBuffer)(void);

Input parameters—NA

Output parameters—NA

Description—The function is used with buffered hardware interfaces. In the LCD control layer call this function in case that the last write word operation was last for the current write sequence.

unsigned char (*D4DLCDHW_DeInit)(void);

Input parameters—NA

Output parameters—Result of the deinitialzation process. 0 for false, 1 for success.

Description—The function deinitializes all things of LCD frame buffer peripherals.

3.3.2.4 Low-Level Touch Screen API Functions Prototypes

This part describes all the function prototype needs to run the touch screen low level driver interfaces.

3.3.2.4.1 Touch Screen Control Layer API

unsigned char (*D4DTCH_Init)(void);

Input parameters—NA

Output parameters—Result of initialization process. 0 for false, 1 for success.

Description—The function initializes all the necessary things to run the touch screen hardware and low-level driver software.

unsigned char (*D4DTCH_GetPositionRaw)(unsigned short *TouchPositionX, unsigned short *TouchPositionY);

Input parameters:

- TouchPositionX—Pointer to the result measured value in axis X
- TouchPositionY—Pointer to the result measured value in axis Y

Output parameters—Result of get position operation. 0 for no touch detected, 1 for touch detected

Description—The function finds out the touch screen status is touched or untouched. In case that the touch screen is touched, the function returns over pointers in parameters of the function and the raw measured data of the touch position.

D4D_TOUCHSCREEN_LIMITS* (*D4DTCH_GetRawLimits)(void);

Input parameters—NA

Output parameters—Pointer to touch screen hardware limits structure

Description—The function returns the pointer on the hardware touch screen limits structure. The touch screen limits structure D4D_TOUCHSCREEN_LIMITS type is defined in d4d_lldapi.h file.

unsigned char (*D4DTCH_DeInit)(void);

Input parameters—NA

Output parameters—Result of deinitialization process. 0 for false, 1 for success.

Description—The function must deinitializes all things initialized by the touch screen hardware and the low-level driver software.

3.3.2.4.2 Touch Screen Hardware Interface Layer API

This part contains function prototypes that are used to interface between the touch screen control layer and the hardware interface layer.

unsigned char (*D4DTCHHW_Init)(void);

Input parameters—NA

Output parameters—Result of initialization process. 0 for false, 1 for success.

Description—The function initializes all the necessary things to run the touch screen interface hardware.

unsigned short (*D4DTCHHW_ReadTouchAxis)(D4DTCHHW_PINS pinId);

Input parameters:

- pinId—Pin identification

Output parameters—Read analog value from the selected pin

Description—The function reads the analog value on the selected pin.

D4D_TOUCHSCREEN_LIMITS* (*D4DTCHHW_GetRawLimits)(void);

Input parameters—NA

Output parameters—Pointer to the touch screen hardware limits structure

Description—The function returns the pointer on the hardware touch screen limits structure. The touch screen limits structure D4D_TOUCHSCREEN_LIMITS type is defined in d4d_lldapi.h file.

unsigned char (*D4DTCHHW_PinCtl)(D4DTCHHW_PINS pinId, D4DHW_PIN_STATE setState);

Input parameters:

- pinId—Pin identification
- setState—Action to do with the selected pin

Output parameters—For the read pin operation return pin value

Description—The function is to control pins that are needed for touch screen control.

unsigned char (*D4DTCHHW_DeInit)(void);

Input parameters—NA

Output parameters—Result of a deinitialization process. 0 for false, 1 for success.

Description—The function deinitializes all things of touch screen interface hardware.

3.4 eGUI/D4D Low Level Drivers Templates

The eGUI/D4D driver contains a template driver for both layers of LCD and touch screen devices. These templates are placed in a low_level_drivers directory in main D4D driver file structure. The templates contain all the necessary things needed to run the driver.

The templates are set of three files:

- d4dxxx_template.c—Mandatory c file that contains mainly filled API structure and all power functions
- d4dxxx_template.h—Optional header file for driver source file
- d4dxxx_template_cfg.h—Optional configuration header file

The “xxx“ in the name of files mark the layer and type of the driver:

- d4dlcd_ —LCD controller layer
- d4dlcdhw_ —LCD hardware communication layer
- d4dtch_ —Touch screen control layer
- d4dtchhw —Touch screen hardware interface layer

3.4.1 How to Use Templates to Create a New User Low Level Driver

Creating a new low level driver from the template is simple and can be executed in a few simple steps.

3.4.1.1 Name of New Driver

First, the new driver has to be created with a unique name that describes the driver and has not already been used in the eGUI/D4D.

Example—For LCD controller layer and LCD driver SSD1926 from Solomon Systech company, the name must be “ssd1926“.

3.4.1.2 New Driver Directory

Second, create a folder for a new driver in the correct place in the D4D directory structure:

Directory paths for drivers:

- LCD controller layer—D4D\low_level_drivers\LCD\lcd_controllers_drivers\
- LCD hardware communication layer—D4D\low_level_drivers\LCD\lcd_hw_interface\
- Touch screen control layer—D4D\low_level_drivers\touch_screen\touch_screen_drivers\
- Touch screen hardware interface layer—D4D\low_level_drivers\touch_screen\touch_screen_hw_interface\

Example—For the new SSD1926 LCD controller a folder must be created, “ssd1926” in D4D\low_level_drivers\LCD\lcd_controllers_drivers\.

3.4.1.3 Copy Template Files

In this step the directory must be filled with the template driver files. The templates are stored in the low-level driver directory and the folder name is called “template”. After copying these files into the user new driver folder, all the files must be renamed to a valid name. This must be executed by replacing the template word in the name of the files by the user name selected in first place.

Example—Copy the content of the template directory placed in D4D\low_level_drivers\LCD\lcd_controllers_drivers\ into the new folder “ssd1926” and rename all the file names with the word “template” to “ssd1926”.

3.4.1.4 Add Driver Files Into eGUI/D4D Project

Finally add these files into the CodeWarrior project. This can be executed by dragging and dropping the new created folder with the renamed template files into the corresponding group in the eGUI/D4D main group.

Example—Drag and drop the new created folder “ssd1926” with the renamed files and place them into the CodeWarrior project with eGUI/D4D to the correct group: D4D\low_level_drivers\LCD\lcd_controllers_drivers\. See figure Figure 3-27.

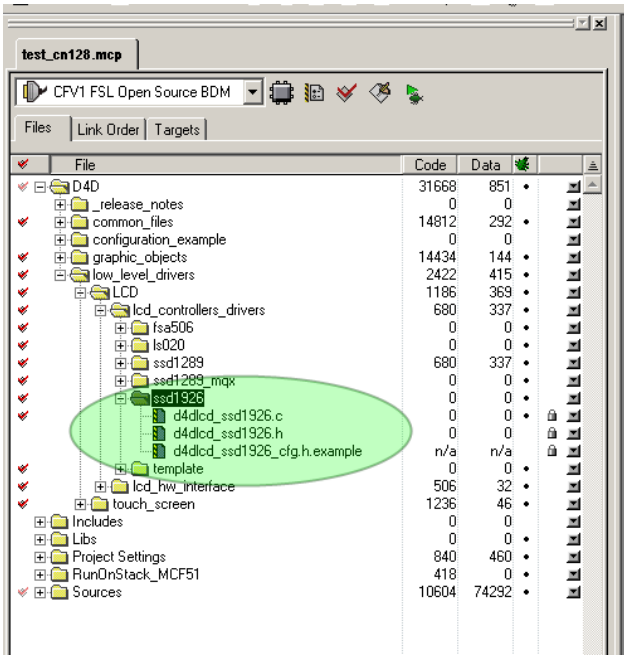


Figure 3-27. New added driver

3.4.1.5 Update File Template Contents to Correct Names

The new driver files from the template contain all the necessary functions and mainly the API structure, but with a default name. All functions, macros, variables, and defines contain the word “template” in a different letter case. The step is renamed and all the contents to their correct names as the driver name selected in the first step.The fastest way to rename all the things needed in these new files are the following:

- Close all editor windows (for example in the CodeWarrior Window menu):

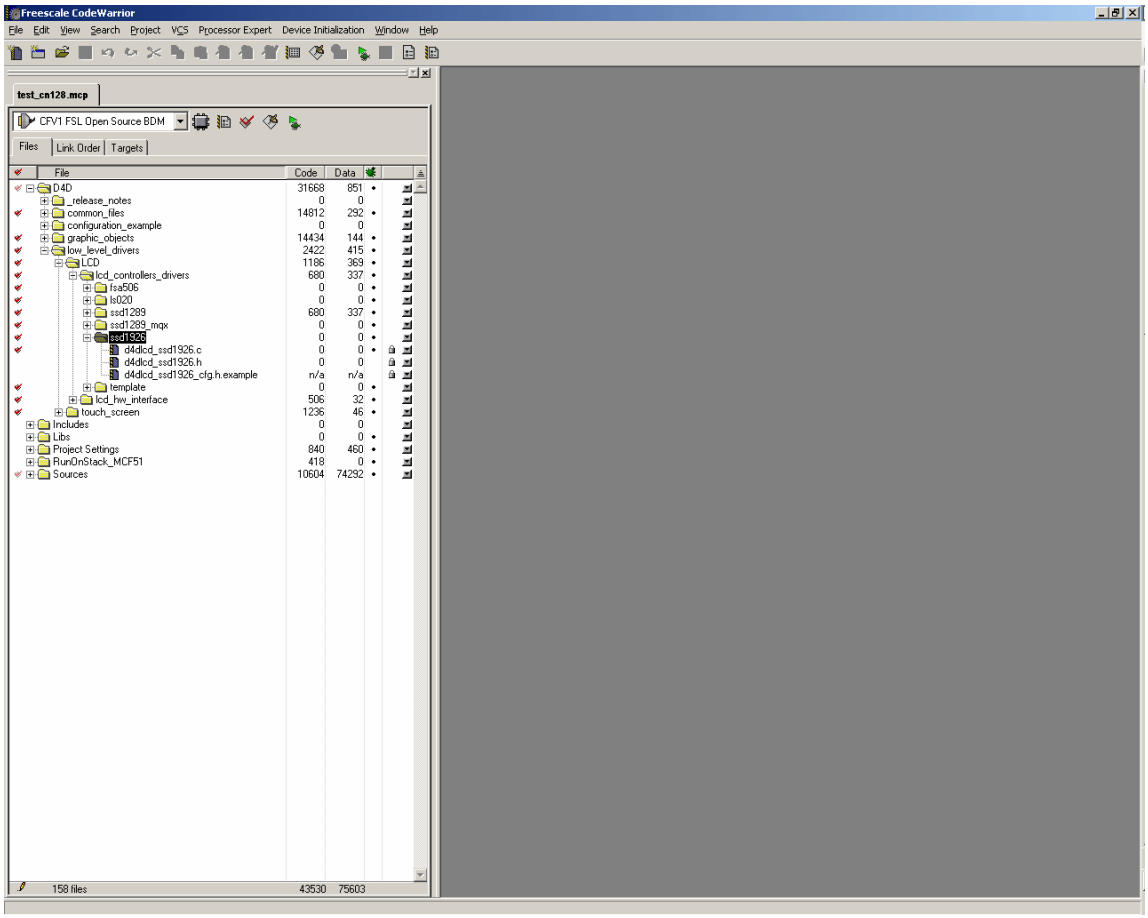


Figure 3-28. CW without any editor windows

- Open all three new driver files to have a better selection in the **Find In Files** menu:

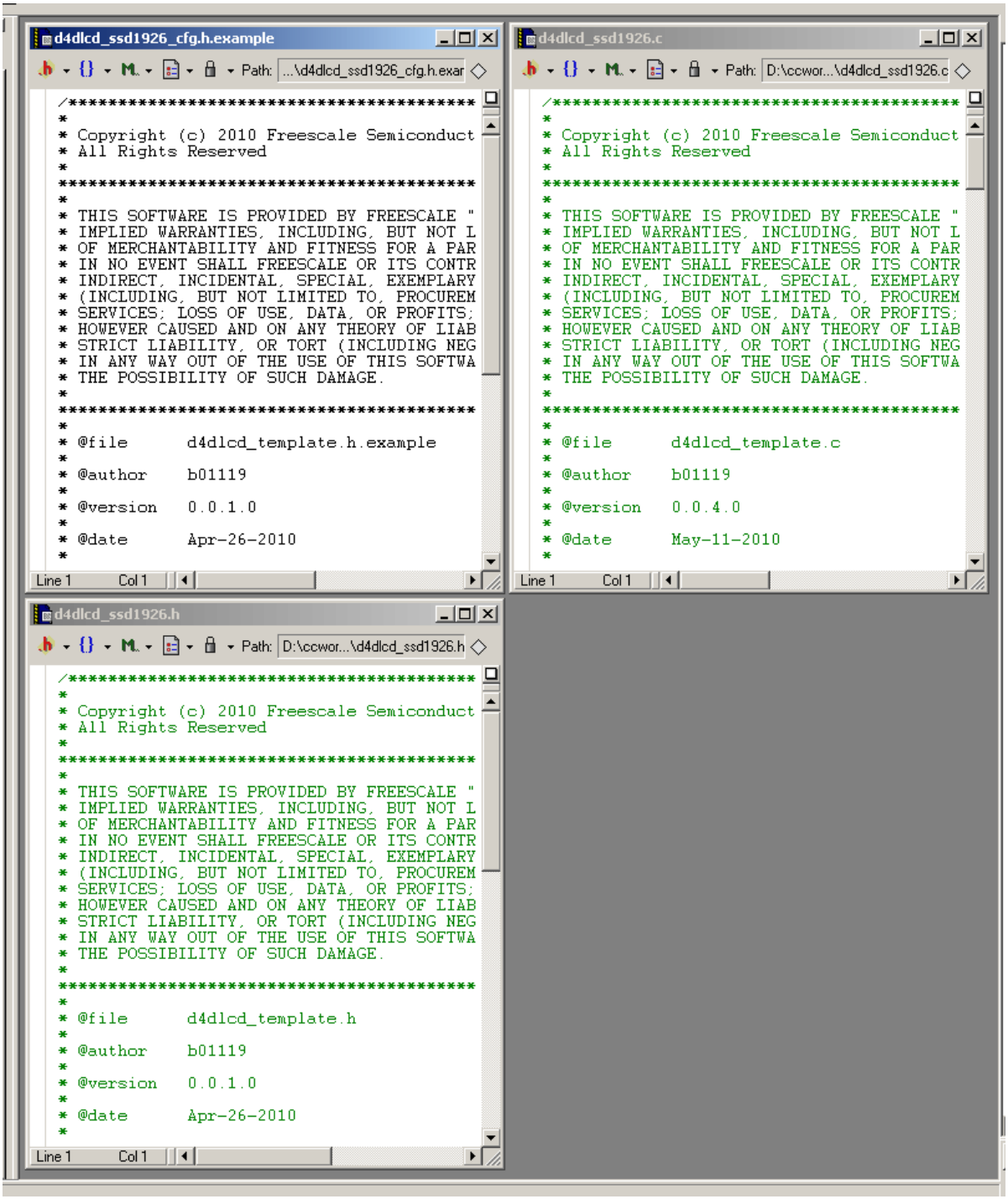


Figure 3-29. New driver open files in editor window

- Open **Find In Files** utility in CodeWarrior search menu.
- Select tab **In Files**
- Set **File Set** to open the **Editor Files**
- Check **Case sensitive** option

- To the **Find** field write the “**template**” string (all small letters), then to **Replace with** field place the new driver name (in this case “**ssd1926**”— all small letters) and click the **Replace All** button

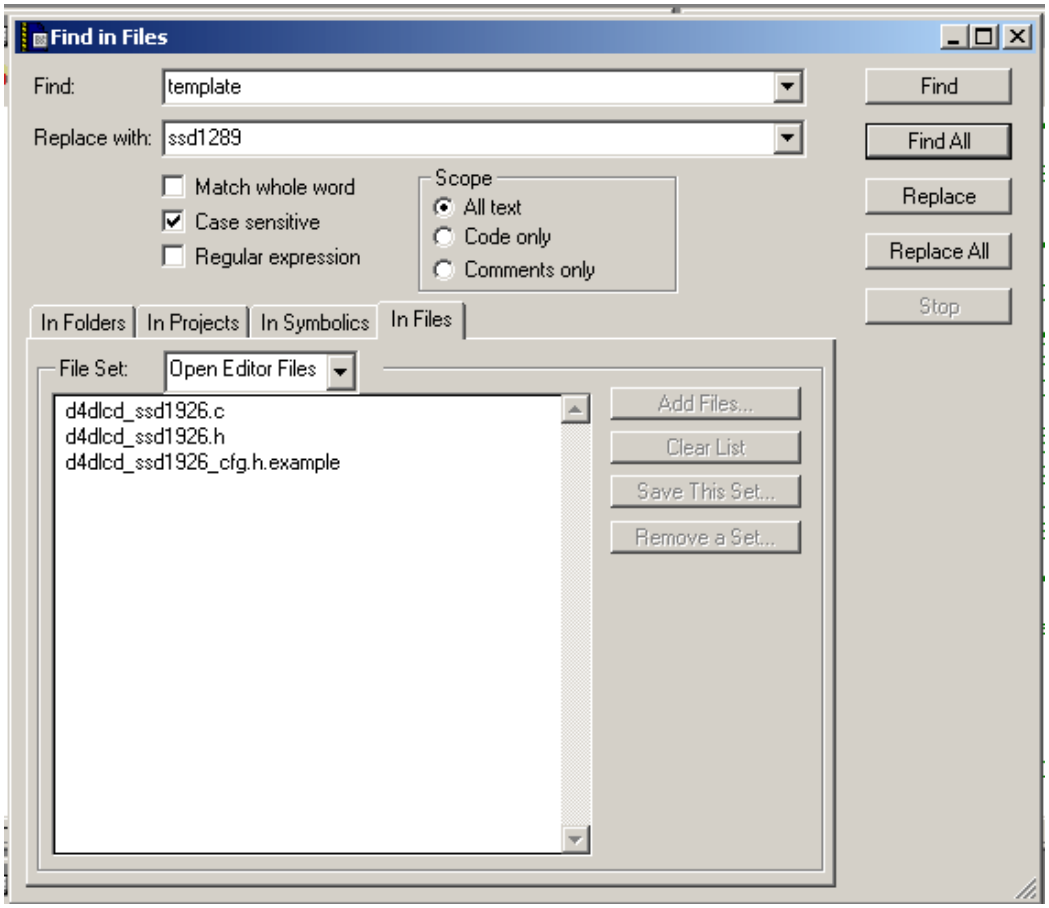


Figure 3-30. Replacing “template” string

NOTE

The **template** is used in comments, the name of the API structure, and in other cases.

- The next string to rename is **Template** with the first letter in upper case. The new string must be the same as in the previous case but with the first letter in upper case or something similar, but there must be a difference from the previous name which is in all lower case letters.
- The last string to rename is **TEMPLATE** with all the letters in upper case. The new string must be the same as the previous case but with all upper case letters. The last upper case letter strings are used for only standard defines in the header files.

3.4.1.6 Fill Up the Function Bodies

The last step is filling up the function bodies with correct code that manages the LCD display for a correct run. After the API is completed by the template the user must fill up only the power code and not the API. This speeds up the new driver development.

3.5 Tips and Tricks

This section has tips and tricks that are good to know for programing with an eGUI/D4D library.

3.5.1 Warning Number C4443 —Undefined Macro

The D4D library low drivers structure invokes this warning (Undefined macro'<MacroName>' is taken as 0) due to a better flexibility of adding new drivers. There are two solutions for avoiding this warning message:

- Disable the warning message in the compiler settings
- Remove from the project all unused low level drivers and keep only the used low level drivers needed to run the application.