

BAKKALAUREATSARBEIT

**Efficient GUI programming using
“Smart-Displays”**

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Bakkalaureus der Technischen Informatik

unter der Leitung von

Univ.Ass. Dipl.-Ing. Alexander Kössler
Institut für Technische Informatik 182

durchgeführt von

Patrick Knöbel
Matr.-Nr. 0925219
G. Dieselstrasse 26 2620 Neunkirchen

Wien, im September 2012

.....

Efficient GUI programming using “Smart-Displays”

In recent years embedded systems as well as user interfaces of embedded systems increased in complexity significantly. To handle the increased demands, graphical touch sensitive displays are often used. Controlling such displays can use a significant amount of resources of the embedded systems.

“Smart-Displays” can be used to avoid the problem of restricting the resources of the embedded system as they provide powerful, high-level graphic functions as well as programmability. Nevertheless, the extra effort required to program such displays is a huge drawback.

The goal of this thesis was to create an environment, which significantly reduces the development time required when using “Smart-Displays” in embedded systems. This was achieved by developing a graphical user interface (GUI) builder platform which consists of a generic firmware for the smart displays and a designer application that allows the user to develop GUIs time efficiently.

Contents

1. Introduction	1
1.1. Problem Statement	1
1.2. Goals	1
1.2.1. Display Platform	1
1.2.2. Display Firmware	2
1.2.3. Designer Tool	2
1.3. Structure of the Thesis	2
2. Concepts	3
2.1. Graphic Display Drivers	3
2.1.1. Dual Ported RAM Display Driver	3
2.1.2. Advanced Display Driver	4
2.1.3. Smart Display Driver	4
2.2. Grapic Display Platforms	5
2.2.1. CFAF240320K-024T-TS	6
2.2.2. Smart GPU	7
2.2.3. ezLCD-301	7
2.2.4. μ LCD-28PT	9
2.2.5. Comparison	10
3. PICASO-GFX2	11
3.1. Extensible Virtual Engine (EVE) Core	12
3.1.1. Memory Organisation	12
3.1.2. Internal Functions	14
3.1.3. Specifications and Performance	14
3.2. 4D Systems Workshop	15
4. Implementation	17
4.1. Overview	17
4.2. Display Firmware	18
4.2.1. Principle Structure	18
4.2.2. 4DGL Files and Includes	18
4.2.3. Render Algorithm	20
4.2.4. Extensible Markup Language (XML) Parser	21
4.2.5. Caching System	22
4.2.6. Element Interface	22

4.3. GUI Designer Tool	25
4.3.1. Class Structure	25
4.3.2. Platform Interface	26
4.3.3. Element Interface	27
5. Results and Conclusion	29
5.1. Results	29
5.2. Conclusion	30
Bibliography	31
A. User Guide	32
A.1. Preparation	32
A.1.1. Display Preparation	32
A.1.2. Micro SD-Card Preparation	34
A.2. Start Designer	35
A.3. Menu	35
A.3.1. Project	35
A.3.2. Export	36
A.4. Tab Settings	37
A.4.1. Basic Setting	37
A.4.2. Other Settings	38
A.5. Tab Resources	38
A.5.1. Resource Types	38
A.5.2. Manage Resources	39
A.5.3. Resource Settings	40
A.6. Tab Events	40
A.6.1. Manage Events	40
A.6.2. Event Settings	40
A.7. Tab Screens	41
A.7.1. Manage Screens	41
A.7.2. Screens Settings	42
A.7.3. Designer Panel	43
A.8. Screen Elements	45
A.8.1. General Properties	45
A.8.2. Shape	47
A.8.3. Image	48
A.8.4. Text	49
A.8.5. Button	50
A.8.6. Checkbox	50
A.8.7. Progressbar	51
A.8.8. Trackbar	52
A.8.9. Tabpage	52

A.9. Image Scaler	53
A.10. Application Interface	54
A.10.1. Events	54
A.10.2. Read Resource	55
A.10.3. Write Resource	56
A.10.4. Special Resources	56
B. Display Design File	57

1. Introduction

The idea for this thesis was born during the preparations for the microcontroller lecture at the Vienna University of Technology. In this course students learn how to program and use microcontrollers. To make the lecture more interesting, the idea came up to use graphic displays as an interface to the controller.

Today nearly every electrical device has a display to show information and to enable an easier configuration. It is getting more and more important, that the use of an interface is as simple as possible. A good example for that trend are mobile phones. Over the last 10 years the functionality has increased extremely, but the interface got more and more simple. Amongst others this was accomplished with the help of touch sensitive graphical displays.

1.1. Problem Statement

Normally the design and the implication of a graphic display into an embedded application takes a lot of time and effort. Furthermore, there are a lot of different types of displays with different interfaces available on the market and it is extensive to choose one.

One goal of this thesis is to reduce the effort of this task. Therefore, a display platform is chosen and a GUI builder is developed. The main focus of this work is to achieve a system, which is very easy to integrate in embedded applications and reduce the development effort for new interfaces.

1.2. Goals

1.2.1. Display Platform

As there are many different types of graphic display platforms and it is not feasible to make a GUI builder, which is compatible to more than one display platform, the first goal is to evaluate available platforms regarding their usability for the task. Therefore, different platforms should be compared and the best should be chosen.

1.2.2. Display Firmware

For the chosen platform a firmware should be developed to run a GUI, which is designed by the GUI builder. The program should have the following specifications:

Scalable: If possible it should be ported to other displays of the same platform easily.

Simple Application Interface: The interface between the display and the application should be as simple as possible.

Touch: The touch function of the graphical display should be fully integrated into the firmware.

Images: It should be possible to display images.

Ready to use GUI elements: There should be predefined GUI elements like buttons, checkboxes, textboxes, etc ...

Extensibility: It should be possible to add new elements to the program and therefore, to the GUI builder.

1.2.3. Designer Tool

The GUI builder, in this thesis called *designer tool*, should provide an environment where one can design the GUI for the display and export it to the display easily. Furthermore, the designer tool should be *cross platform compatible* to increase the usability.

1.3. Structure of the Thesis

Chapter 2 gives an overview about the different display platforms evaluated and it will be established, which platform has been chosen and why. In Chapter 3 a detailed description of the chosen platform is provided. In Chapter 4 the whole practical work is presented. Therefore, all design decision are documented. In Chapter 5 the whole project is summed up and the results are discussed.

In Appendix A one can find a user guide for the designer tool application. Furthermore, in Appendix B the low level design definition file is described.

2. Concepts

This chapter gives an overview about the different ways controlling a graphic display. Furthermore, some suitable displays for the thesis are evaluated.

2.1. Graphic Display Drivers

A display is an electrical device to visualize information. As a “segment display” is a simple display, which only can show alphanumeric characters, in this thesis a “graphic display” is used as one can visualize any kind of information, from simple lines to very complex 3D graphics there only.

One of the main problems using graphic displays, is the high complexity to control them. To reduce this complexity, graphic display drivers are used. As there are different types of display drivers, in this section some of them are described.

2.1.1. Dual Ported RAM Display Driver

The simplest concept to control a graphic display is shown in Figure 2.1.

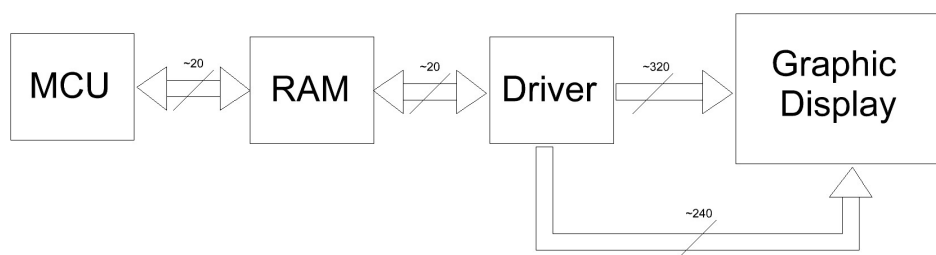


Figure 2.1.: Dual ported RAM graphic driver schematic

In this case a very simple driver is used for controlling the display. The centrepiece is a dual ported random access memory (RAM). Each cell of the RAM is dedicated to a pixel of the display. Dependent on the number of different colors available a cell can have a capacity up to 32 bit.

This driver updates the display with the data from the RAM cyclically. The microcontroller (MCU) has to fill the RAM with the data, which should be displayed. Therefore, the MCU needs high calculation effort to generate the needed graphical data.

For example if you want to draw a line on the display, the MCU has to calculate the line segments and write each segment into the RAM.

2.1.2. Advanced Display Driver

Figure 2.2 shows an advanced driver. The main difference to the dual ported RAM driver is the RAM access method. Now it is only possible to access the RAM over the driver interface.

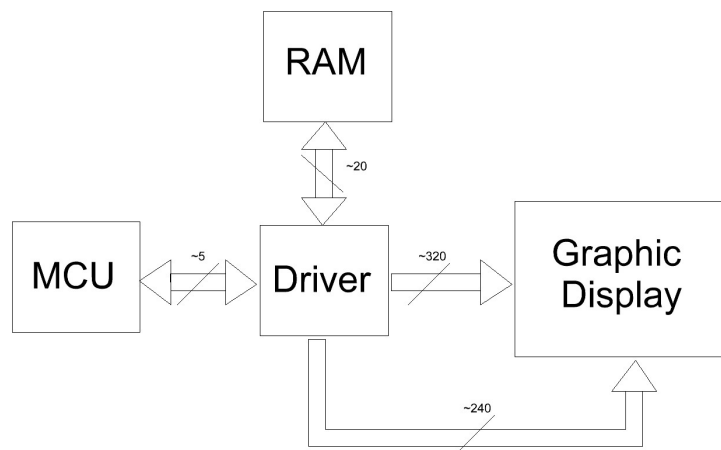


Figure 2.2.: Advanced graphic driver schematic

The MCU communicates with the display driver via a serial interface. It is still possible to manipulate the RAM cells directly, but the driver also provides high-level functions, e.g., if you want to draw a line it is sufficient to define the start and the end points only. Some drivers even provide high-level functions to plot circles and other geometric figures.

This extra functionality of the interface significantly reducing the necessary computing time of the MCU.

2.1.3. Smart Display Driver

The structure of the smart display drivers are very similar to the advanced drivers. Figure 2.3 shows the schematic.

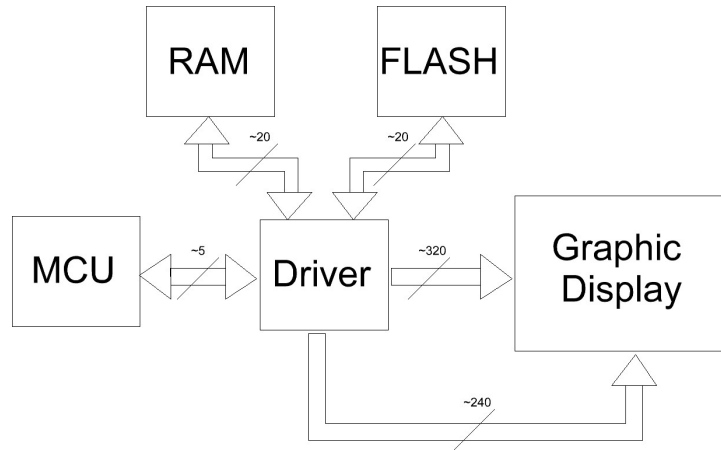


Figure 2.3.: Intelligent graphic driver schematic

One difference between the advanced display drivers and the smart display drivers is that latter are configurable. For example, it is possible to save images or animations on the flash memory and directly play them without further MCU assistance.

Normally the interface provides more high-level functions than an advanced display driver. For example, some drivers provide functions for GUI elements like a button that are ready to use.

The goal of all these extra functions is to reduce the communication between the MCU and the driver as well as reduce the workload of the MCU. There are also some drivers like the $\mu LCD-28PT$ investigated later in detail, which can store and execute programs by them-self. Sure that nearly all computationally intensive graphic tasks are done by the display driver. This reduces the development time of the embedded software.

2.2. Grapic Display Platforms

Since there are many different displays on the market, we made the following restrictions to fix some minimal requirements:

- The displays diagonal must be between 2" and 3" inches.
- It must be a full color graphical display with a minimum resolution of QVGA (320x240).
- The price of the display should be lower than 100€ for a single piece.
- A touch panel should be included.

In the following some display modules that reach the requirements are introduced. Clearly, there are a lot more displays with similar characteristics as listed below. Therefore, primary the cheaper display platforms, which have the highest availability on the inner-European marked, are picked.

2.2.1. CFAF240320K-024T-TS



Figure 2.4.: Graphic display (CFAF240320K-024T-TS)

This graphical display is highly available on the market, and there are some other displays with identical specification.

Specifications

Diagonal Dimension: 2.4 inch

Resolution: 240 x 320 pixel

Display Technology: TFT

Driver: ILITek ILI9325 (dual ported RAM driver)

Interface: i80 System Interface 18-bit bus

RAM: integrated 172800 bytes

Price: from 5 € to 40 €

Review

This display is very cheap and highly available on the marked and reaches our basic requirements.

The display has only a very basic driver included featuring a dual ported RAM interface as described in Section 2.1.1. Therefore, an extra processor is needed to develop a GUI builder platform for it.

2.2.2. Smart GPU



Figure 2.5.: Graphic display (Smart GPU)

Specifications

Diagonal Dimension: 2.4 inch

Resolution: 240 x 320 pixel

Driver: Smart display driver ¹

The following functionalities are integrated:

- Text rendering
- Load images from SD-Card
- Draw geometric primitives
- Handle touch

Interface: Serial (RS-232 3.3V TTL)

FLASH: Micro SDCard up to 32 GB

Price: 70 €

Review

The price of the display is in the range and the extra functionalities are very useful. But an extra processor board is still needed.

2.2.3. ezLCD-301

Specifications

Diagonal Dimension: 2.6 inch

¹For more information read Section 2.3.

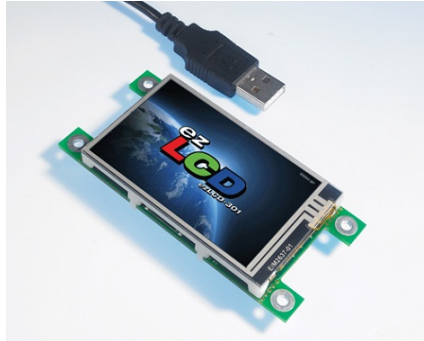


Figure 2.6.: Graphic display (ezLCD-301)

Resolution: 240 x 400 pixel

Driver: Smart display driver ²

The following functionalities are integrated:

- Text rendering
- Font management
- Touch management
- Load images from FLASH
- Draw geometric primitives
- High level function for many GUI elements
- Primitive macros

Interface: USB³, Serial (TTL)

FLASH: Integrated 4 MB

Price: 80 €

Review

The display has many very useful high-level functions integrated, especially the included management for GUI elements. It is also possible to define macros to reduce and simplify the communication.

The need for an extra processor board is still given. But a very basic MCU should reach the given requirements.

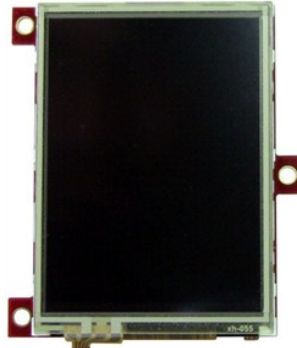


Figure 2.7.: Graphic display (μ LCD-28PT)

2.2.4. μ LCD-28PT

Specifications

Diagonal Dimension: 2.8 inch

Resolution: 240 x 320 pixel

Driver: PICASO-GFX2 smart display driver⁴

The following functionalities are integrated:

- Text rendering
- Font management
- Touch management
- Load images from SD-Card
- Load Videos/Animation form SD-Card
- Load Sounds form SD-Card
- Draw geometric primitives
- High level function for many GUI elements
- extensible virtual engine (EVE) is integrated (it can run 4D graphics language (4DGL) programs)

Interface: Serial (RS-232 5V TTL)

FLASH: Micro SD-Card

Price: 58€

²For more information read Section 2.3.

³The USB port is used to program the FLASH.

⁴For more information read Section 2.3.

Review

The big advantage of this platform is that it is possible to execute a real program on the display driver. Therefore, the need for an extra processor board is not given.

2.2.5. Comparison

The most suitable display for the thesis is the μ LCD-28PT. The main reason therefore, is the possibility to execute programs directly on the driver. So there is no need for the development of an extra processor board. Also, the price of the μ LCD-28PT is the lowest, of all “smart displays”.

3. PICASO-GFX2

The chosen display $\mu LCD-28PT$ is based on the 4DGL graphics controller (PICASO-GFX2). Figure 3.1 shows an overview of the components build into the controller.

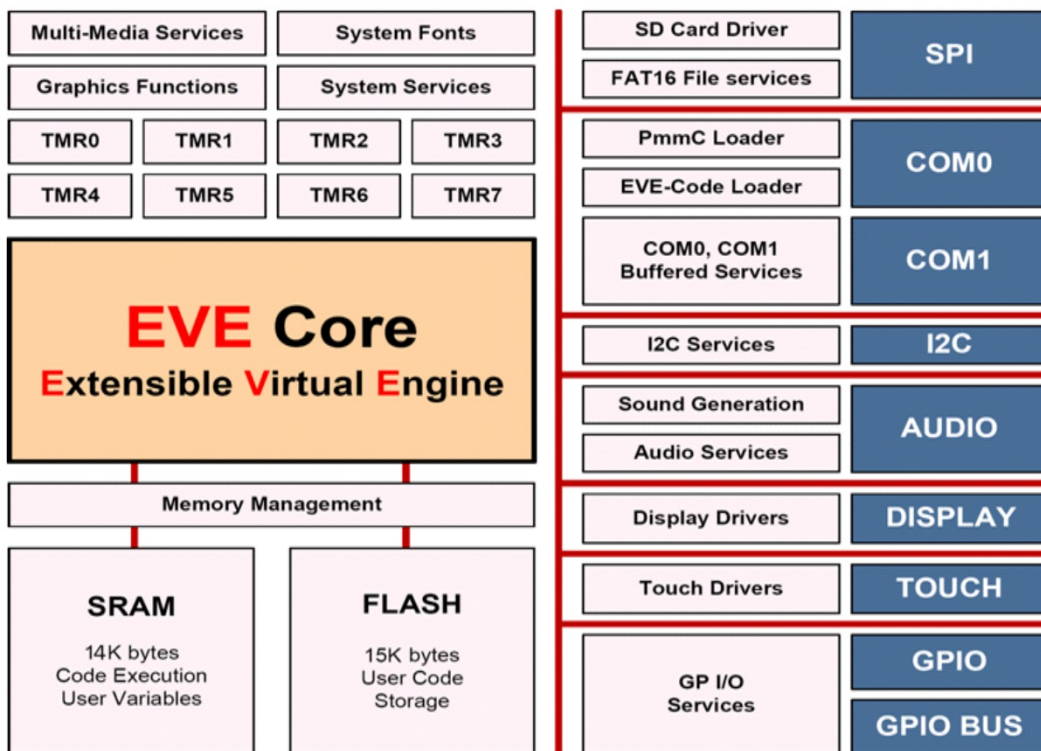


Figure 3.1.: PICASO-GFX2 component overview

*“The **PICASO-GFX2** is a custom embedded 4DGL graphics controller designed to interface with many popular OLED and LCD display panels. Powerful graphics, text, image, animation and countless more features are built right inside the chip. It offers a simple plug-n-play interface to many 16-bit 80-Series colour LCD and OLED displays.”* [Lab12b, page 2.]

3.1. Extensible Virtual Engine (EVE) Core

4D Systems have developed their own programming language called *4D graphics language* (4DGL). It is a mixture out of *Basic*, *Pascal* and *C*. The language is compiled to a byte code, which is then executed by the EVE core.

The following code is a *hello word* example written in 4DGL:

```
func main()  
    print(" Hello World!");  
endfunc
```

There are often updates available for the display driver. On the homepage [Sys] one can download the newest firmware and flash it on the driver with the tool called *PmmC Loader*.

In the following sections the performance and architecture of the 4DGL is evaluated.

3.1.1. Memory Organisation

The EVE core has access to three different memory sources:

SRAM: The static random access memory (SRAM) is used for variables and executable byte code. It is the fastest memory source and embedded into the controller.

FLASH: The FLASH memory is used to store byte codes and constants. This memory can only be written by the “4D Workshop”, that is the official *integrated development environment* (IDE) for the 4D Systems displays and is also integrated into the controller.

SD-Card: It is possible to attach a micro SD-Card to the diver. The SD-Card is used to save images and byte code, but it is not possible to execute the byte code directly from the SD-Card. To execute the byte code stored on the SD-Card one has to copy it into the SRAM first.

Start up

Usually on start-up the program from the FLASH memory is copied to the SRAM, because the execution speed from SRAM is up to two times faster. Clearly this reduces the memory available for variables. To prevent this behaviour one can add the directive “#MODE RUNFLASH” at the beginning of the 4DGL code enforcing code execution directly from the FLASH memory.

Stack and Heap

The SRAM is separated in two parts, the stack and the heap. The stack is used for local variables and functions calls. The size of the stack can be defined by a directive called `STACK`. The heap can be used to allocate memory. When required on start up the byte code of the main program and also the executable sub programs from the SD-Card are loaded into the heap. The heap management is very primitive and it is recommended to free the memory in the same order as it was allocated, otherwise one will end up with a highly fragmented memory.

Functions, Variables and Indirect Memory Addressing

The variable and pointer arithmetic of the 4DGL is very primitive. The only available variable type is integer and there is also no explicit type for variable and function pointers. Like in C, it is possible to get the address of a variable by using the “&” operator. To get the address of a function just use the name of the function as a variable. The “*” operator can be used to make an indirect memory access.

There are two different address spaces. One is used for constants and the other for variables. It is not documented how the EVE core differentiate between them. The problem is that the main program and all sub programs only share a common variable space. If you pass a pointer, which pointing to constant space, to a sub program, the sub program will dereference the pointer with it own constant space and therefore, one will end up in an undefined behaviour.

A sub program, similar to the main program, can include multiple functions. Over the integrated function `file_LoadFunction("file name")` it is possible to load sub programs from the SD-Card into the heap. The function returns the pointer to the main function of the sub program. This is the only way to get a function pointer, which points on the variable space.

To support string operation the internal function `str_Ptr(&var)` is used to convert an integer pointer into a byte pointer, to address single charaters of a string. It is not documented how this conversion works and it is also not possible to use the “*” operator on byte pointers to get a character value. So the byte pointer only works with string functions. The most string functions also have problems to differentiate between byte pointers and constant strings. Therefore, you have to test which function supports which kind of combination.

3.1.2. Internal Functions

There are hundreds of different internal functions provided. For more information read the manual [Lab12c]. The following sectors are covered:

GPIO: These functions are used to control the general purpose input/output (GPIO) ports.

Maths: The platform provides basic mathematical functions like: abs, min, max, sin, cos, sqrt ...

Strings: The string functions are used to manipulate strings and to find sequences in strings.

Graphical: These functions are used to draw basic figures like: lines, rectangles, circles ...

SD-Card: These function are used to load or display content from the SD-Card.

Serial Communication: The Platform provides two serial ports and these functions are used to control them.

I2C Bus: Basic functions are provided to drive an inter integrated circuit (I2C) bus.

Timers: The provided time functions can be used to measure time and periodically call functions.

Sounds: To play sounds from the SD-Card functions are provided.

Touch: The touch screen support is fully integrated into the platform. There are function provided to react on touch activities.

Memory: Here are basic functions provided to dynamically allocate memory.

3.1.3. Specifications and Performance

There is no documentation about the performance and the structure of the EVE byte code and EVE core. Here are some additional information about the display driver, which are determined by empirical tests:

SRAM Size: 14818 bytes

Maximal Stack Size: 8192 bytes (shared with the SRAM)

FLASH: 14400 bytes

Render speed:

For the speed test figures were rendered with random dimension on an area of 240 x 240 pixel.

Lines: 2500 figures/s

Triangles: 1000 figures/s

Filled Triangles: 220 figures/s

Rectangles: 5000 figures/s

Filled Rectangles: 700 figures/s

Circles: 1100 figures/s

Filled Circles: 260 figures/s

Load Sub Programs:

Load: 5 μ s/byte of the byte code

Memory Usage: byte code + 600¹ bytes

3.2. 4D Systems Workshop

4D Systems provides some tools for the development. Figure 3.2 shows the 4D Systems Workshop, the main application for development of 4DGL codes.

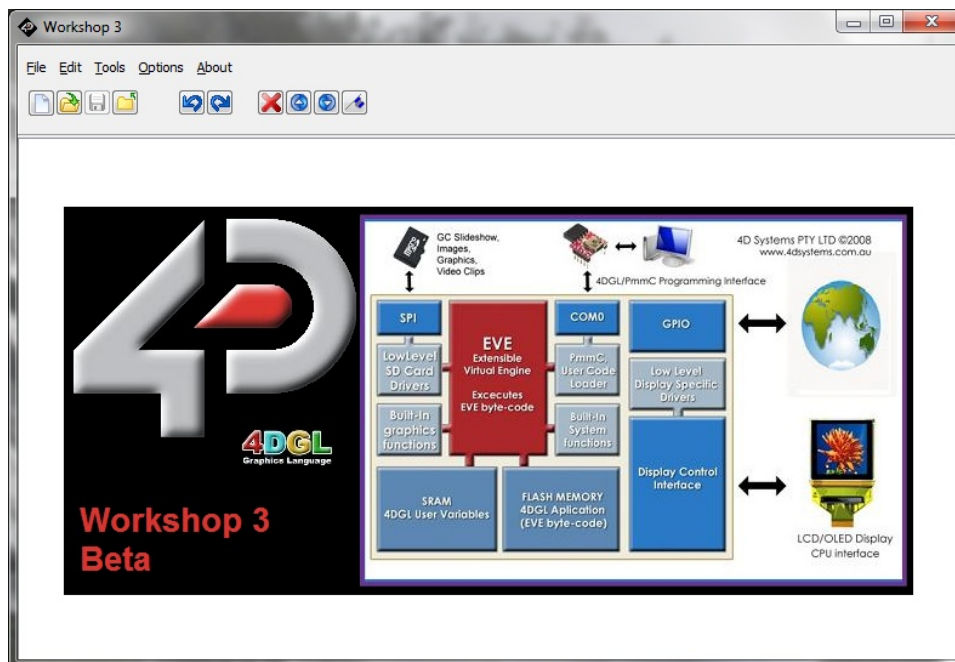


Figure 3.2.: 4D Systems Workshop

The following tools are provided:

¹The internal function `file_LoadFunction` uses some extra space to load the sub program.

4D Workshop: It is the main integrated development environment (IDE) for 4DGL development. It includes the compiler and the possibility to flash the programs to the display.

PmmC Loader: This tool is used to update the display driver firmware.

Font to Raw Bitmap Converter: With this tool it is possible to prepare a new font for the usage with 4DGL.

RMPET Software Tool: The SD-Card can be prepared for the usage with this tool.

Graphics-Composer Software Tool: This tool is necessary to convert images to the format with is supported by the display driver.

All provided applications can be executed on windows platforms only.

4. Implementation

In this chapter the structure of the developed software is described. It splits into the firmware part and the GUI designer part. The main interfaces are specified as well.

4.1. Overview

Two programs were developed. One for the display and the other one for the user who designs the GUI for the display.

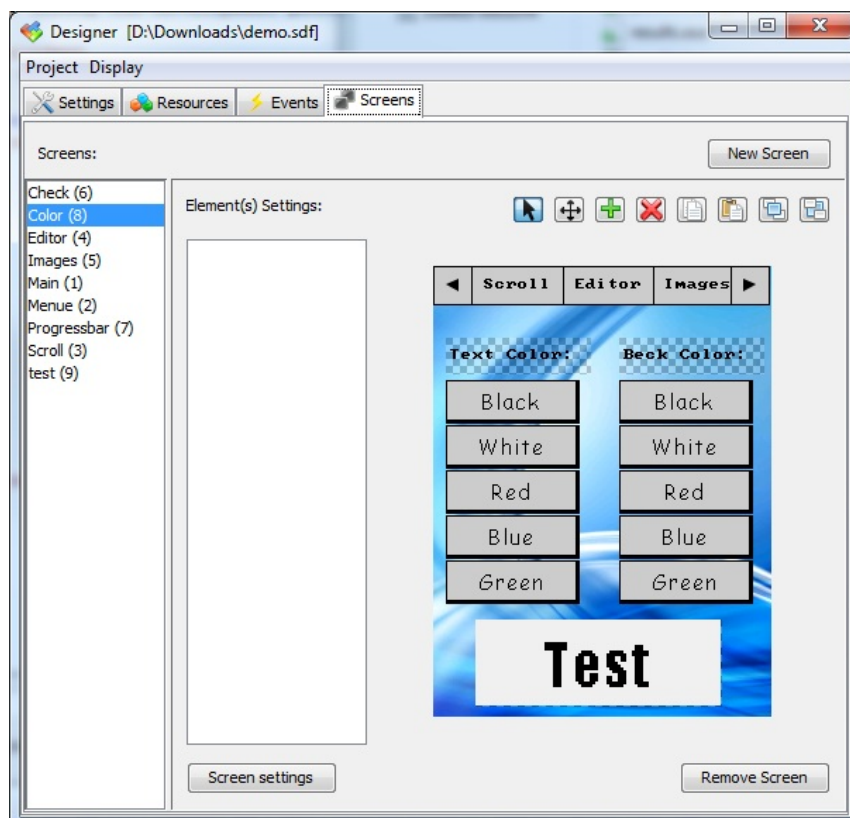


Figure 4.1.: GUI designer tool screen shot

The program running on the display driver, is used to load and display the design information stored on the micro SD-Card connected to the display. This

information is used to display a GUI and to interact with the application connected to the display. Due to the limited flash memory of the display driver the necessary sub programs have to be stored on the SD-Card too.

The second program called GUI designer tool is the GUI builder for the display. Figure 4.1 shows the screen window of this tool, for more information about the usage read Section A. This program helps developers to generate a suitable GUI for their projects. The file “display.xml” describing the GUI is generated by the designer tool and stored on the SD-Card. The designer tool also provides the necessary sub programs needed by the display driver.

For more information about the structure of the “display.xml” file read Section B.

4.2. Display Firmware

The whole firmware is written in 4D graphics language (4DGL). More information about the syntax can be found in [Lab12a].

4.2.1. Principle Structure

Figure 4.2 shows the flow chart of the display program.

The program file “main.4dg” is flashed to the display and directly executed from the flash memory. Due to the limited size of the flash memory many parts of the program are exported to the SD-Card and are loaded on demand into the SRAM. For more information about the different files read the next Section.

4.2.2. 4DGL Files and Includes

In this section all used 4DGL files are described.

Common Used Functions

attribute.inc: These functions are used to manage attributes of an extensible markup language (XML) element.

cache.inc: This file provides a function to create the cache file.

comport.inc: This library is used to manage the communication protocol.

errormessage.inc: This file provides a function to display error messages.

file.inc: This library provides high-level file load and store functions.

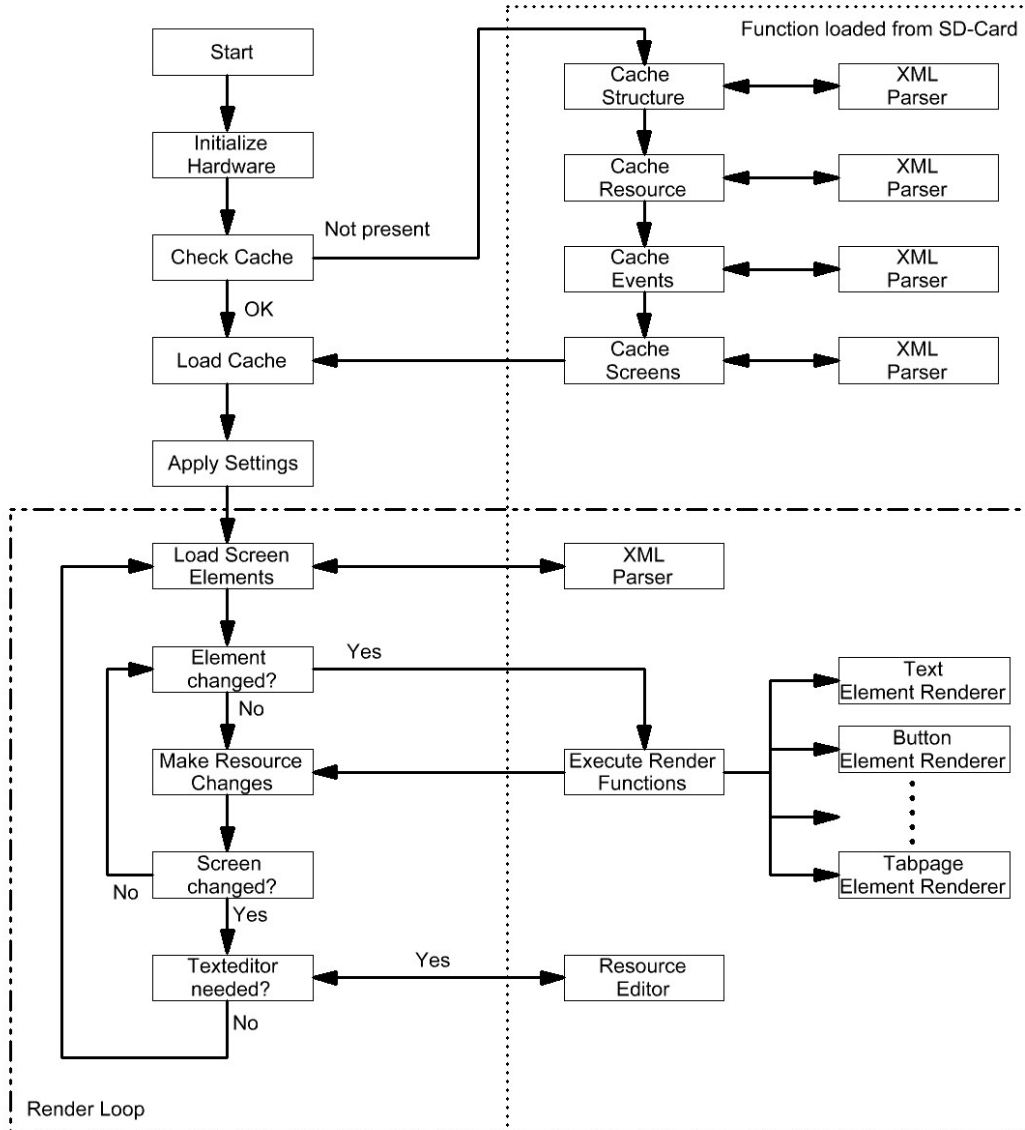


Figure 4.2.: Structure of the display firmware

fontmanager.inc: This library is used to load fonts dynamically from the SD-Card.

loader_functions.inc: This file provides functions to load the cache file.

renderhelper.inc: This library is used for the render programs to share common functions.

resources.inc: Here are some functions provided to handle resources.

string.inc: Some high-level string function are provided.

subfunction.inc: This library is used to load functions dynamically from the SD-Card.

systemconst.inc: In this file all needed constants are defined.

systemdata.inc: In this file all needed constants for the flash memory are defined.

timing.inc: This library can be used to measure time.

xml.inc: This library is used to parse XML files.

Main Program

main.4dg: This 4DGL program is the main entry point. It is flashed to the display. All other programs are stored on the SD-Card.

Sub Programs provided from the SD-Card

error.4dg: This program displays an error screen and stops the whole execution. All error messages are coded into this file.

xml.4dg: This is the XML parser. For more information read Section 4.2.4.

cache1.4dg: This program builds the cache file for the root element of the “display.xml” file

cache2.4dg: This file expands the cache file with the events information.

cache3.4dg: This program adds the resource information to the cache file.

cache4.4dg: This file caches the screens.

fontmanager.4dg: The font manager is used to pack a font. As it is not possible to load a font directly from the SD-Card, the program is used.

reschang.4dg: This program is an editor to edit resources.

The other sub programs are used to render elements or fonts.

4.2.3. Render Algorithm

This chapter provides a high-level overview about the render algorithm only describing the mechanism in which way the elements of a screen get updated.

Screen elements are only rerendered if something happened. There are several mechanisms, which can trigger a rerendering:

- A resource used in the element attributes has been changed.
- The element gets paint over by another element.

- A touch event occurs on the element.
- The time out for a rerender occurs.

When an element has to be rerendered, the element render program is loaded and executed. This program can return two different states. One indicates that the element got repainted and the other one indicates that the element is transparent and therefore, the background has to be repainted.

In the first case the algorithm looks for elements, which are overlapping the repainted element. All found elements get marked for rerendering.

In the second case the algorithm repaints the area of the element with the screen background, Then it marks all elements, which are overlapping and which are under the active element. At last the rendering algorithm jumps to the first element in the list which is marked.

If more transparent elements are overlapping each other, it is possible that some elements get repainted more than ones. It would be possible to compensate this problem, but due to the less available memory it is not implemented. For more information about the algorithm read the comments in “main.4dg” file.

4.2.4. Extensible Markup Language (XML) Parser

To interpret the “display.xml” file an XML parser is used. The parser is a complex state machine, which parses each character of the XML file. The following states are used:

E_BODY: This state is active in an element body or outside of the root element. It is also the initial state.

E_OPEN: This state is active after the “<” character, on the beginning of an element or the beginning of an end element.

E_O_NAME: This state is active during the parse of the tag name.

E_A_BODY: In this state the parser is in an element.

E_A_BODY_MASK: This state is used to recognize an end tag (“/>”).

E_A_NAME: This state is active during an attribute name.

E_A_EQUAL: This state is active between the state and the value.

E_A_VALUE: During the Value of the attribute this state is used.

E_A_VALUE_MASK: To mask a “double quote” this state is used.

E_CLOSE: This state is used to recognise an end element tag.

E_C_NAME: This state is used in the name of an end element tag.

The parser is separated into two parts, to get a more clearly arranged code. The first part is the state machine itself, and the second is used to build the data structure. Therefore, the state machine passes events to the second part.

For more information about the XML parser read the comments in the “xml.4dg” file.

4.2.5. Caching System

The parsing of the whole “display.xml” file at start up takes very long. Therefore, a caching system was developed. All information is collected and saved on the SD-Card to the file “XMLCACHE.DAT”. In this chapter the structure of this file is described.

The cache file is base on sections. Table 4.1 shows the length and the description for each section.

On start up the first 6 bytes from the cache file are loaded. If the length and the checksum of the “display.xml” file matches the saved ones, the rest of the cache file is loaded. Therefore, the cached data is used and there is no need to parse the XML file.

For more information about the date structures and the elements read the “systemconst.inc” file. For more information about the caching read the files: “cache.inc”, “cache1.4dg”, “cache2.4dg”, “cache3.4dg”, “cache4.4dg” and “loader_functions.inc”.

4.2.6. Element Interface

It is easily possible to add new elements to the GUI. Therefore, the following steps are necessary:

- Create a render program for the element.
- Add the file name and the specification for the element to the “system-data.inc” file.

Render Program

The render program is stored on the SD-Card and loaded on demand from the main program. It uses the following interface:

The following list describes the parameters hand over to the main function of the render program:

Name	Size in Bytes	Description
File Length	4	The length und checksum is used to check if the XML file has changed.
File Checksum	2	
Root Elements	4* CXML_SIZE *2	This section saves the basic structure of the 4 following elements: “display”, “resources”, “events”, “screens”. This informations are used to load attributes.
RES_CNT	2	The number of the defined resources are stored here.
Resources	RES_CNT* RES_SIZE *2	All important informations of all resources are stored here. The name and default values are links to the resource strings.
RES_LENGTH	2	Here is the length of the resource string stored.
Resource Strings	RES_LENGTH	In this section the strings of all resource names and default values are stored.
EVENT_CNT	2	The important informations about the events are stored here.
Events	EVENT_CNT* EVENT_SIZE* 2	
SCREEN_CNT	2	The important informations about the screens are stored here.
Screens	SCREEN_CNT* SCREEN_SIZE* 2	

Table 4.1.: Structure of the cache file

x, y, width, height: The dimensions of the element are hand over here. They are also available over the element attributes, but to reduce computing time it is recommended to use them.

rerendersource: There are different reasons for an element to get rerendered. This parameter can have the following values:

REREDNERSOURCE_INIT: The render program is called the first time. Therefore, the local variables for the element are not initialised. Also the background is new and transparent elements can be rendered.

REREDNERSOURCE_RENDER: The background is repainted. Therefore, transparent elements can be rendered.

REREDNERSOURCE_RESOURCE: A resource has changed.

REREDNERSOURCE_OVERPAINT: The element was painted over by

an other element. Thus, the background was not repainted.

REREDNERSOURCE_TOUCH: A touch event occurred. It is not necessary to repaint the element.

REREDNERSOURCE_TIME: The time set for reredner expired. It is not necessary to repaint the element.

attributes: Here the pointer to the element attributes is hand over. The attributes pointing to resource are already replaced with the resource string. It is not allowed to manipulate attributes. For more information read the file “attributes.inc”.

locvalues: A pointer to the local variables of the render program. This array exists for the life time of a screen and get lost on a screen switch. The length of this array is specified in the file “systemdata.inc”.

touchstatus, touchx, touchy: Here the touch informations are hand over. It is also possible to get an update here, without having “REREDNERSOURCE_TOUCH” as rerednersource. For more information about the touch status read [Lab12c, section 2.6.19].

&event: This is a return parameter. Here you can return the number of an event or “0” for no event. This event gets triggered by the environment after the end of the program. Also resource manipulations are executed before.

&resName: The name of a resource, which should be changed, is returned by this parameter. If “0” is returned, the next parameter is ignored and no resource gets changed.

&resValue: Here the value for the resource is returned. If “0” is returned an text editor gets opened. The value must be an integer pointer, which has to be allocated in the render program. It is not allowed to pass a string pointer from an attribute.

&nextRender: The time for the next repainting of the element can be specified here. If you pass “-1” the render time of the element will not expire.

Return Value: The return value of the program is also important. Three values are allowed:

ELEMENTRETURN_NONE: The program indicates that the element was not repaint. This is only acceptable when the render-source was “REREDNERSOURCE_TOUCH” or “REREDNERSOURCE_TIME”.

ELEMENTRETURN_TRANSPARENT: Here it is possible for the program to indicate that it has a transparent element to render and the background is not rerendered. This value is not allowed

when the `rendersource` has the following values: “REREDNER-SOURCE_INIT”, “REREDNERSOURCE_RENDER”.

ELEMENTRETURN_OK: This value indicates, that the element has been repainted.

4.3. GUI Designer Tool

The designer is programmed in Java, which is cross platform compatible. For more information about the functionalities of the designer read Section A.

4.3.1. Class Structure

In this section only a short overview about the structure is provided. A detailed documentation about the software will go beyond the scope of this thesis.

In the following list the folders and some important files of the source code are described:

database: In this folder all classes for the database are stored. The database is used to save and load a project. The following classes are important:

Database: This is the main class for the database. It is implemented as a singleton and is also a wrapper class for the private member “Data”. The “Data” class is serializable and holds all information for a project.

DatabaseResource: This class is used to save all resources of the project.

DatabaseEvent: In this class the information for events are saved.

DatabaseScreen: This class is used to save screens.

DatabaseSElement: All properties of an element are handled in this class.

DatabaseSEP...: All classes, which starts with this name, are used to save a property of an element. For all different kinds of properties there exists such a class.

IEventHandler: This is an event interface. The database is based on an event system. Every change made in the database triggers an event. Therefore, all registered classes are notified.

designer: In this folder all forms, panels and dialogues can be found.

display: The “display” folder is the most relevant one for administration. The following classes are imported to mention:

EBasicProperty: In this enumeration all basic properties are listed. They are also linked to the corresponding database entries.

EProperties: Here are all usable properties listed and they are all based on the basic properties.

EElements: This enumeration includes all types of screen elements. It is defined which properties an element has and how it is rendered and exported.

EDisplaySettings: In this enumeration all supported display platforms are listed.

XMLGenerator: This class is used to build the XML file for the display.

icons: In this folder images for the GUI of the designer are stored.

resource: This folder is used for all included files. Here are the 4DGL programs and fonts stored. The “run.bat”¹ script is provided to compile a copy of the needed render programs to this folder.

tools: The classes in this folder provides project independent functionalities.

All files with the ending “properties” are language files. It is possible to add more languages by cloning them. For more information read [Orab].

4.3.2. Platform Interface

The designer tool is designed to be very flexible and expandable. To add a new platform it is only necessary to add a new element to the “EDisplaySettings” enumeration. Therefore, the following functions must be overwritten:

getName(): This function must return the name of the new platform.

getSerialSpeeds(): Here an array of the supported communication speeds is expected to be returned.

getDefaultSpeed(): The “getDefaultSpeed” function must return the default communication speed.

getContrast(): Some displays have no contrast settings. Therefore, this function must return the value “false”.

getResourceMaxSize(): Due to the limited SRAM the buffer size for the communication is also limited. This function should return the size of the buffer.

getXMLMaxAttributSize(): Also the buffer for the attributes of an element is limited. Therefore, this function should return the maximum length of an element note.

¹To use “run.bat” the make environment must be installed.

getDimension(): At last this function has to return the resolution for the new display.

4.3.3. Element Interface

This section describes how to add a new screen element. The screen element is based on properties and a property again is based on a basic property.

Add a Basic Property

The first step is to define all needed basic properties. Normally all needed basic properties are available. Adding a new one is quite complicated.

The following steps have to be done:

- Add a new class in the database which includes the interface “ADatabaseSEProperty” to save the data for the basic property.
- Also a dialogue window must be added to allow the user to change the property.
- At last a new entry in the “EBasicProperty” enumeration must be added.

WARNING: The property passed by the environment can be “null”, which means that no settings are present.

Add a Property

To add a property it is only necessary to add a new entry to the “EProperties” enumeration. If possible, use existing properties. Therefore, the user gets the possibility to change the property form different elements with different types simultaneously.

Add an Element

To add a screen element you must add a new entry to the “EElements” enumeration. Therefore, the following informations and functions have to be provided:

Enumeration Item Name: The name of the item is used as type name for the export. Therefore, it must correspond with the definition in the display program.

File name: The name of the render program must be defined. Also the render program must be copied to the resource folder.

Properties: All properties necessary to represent the element have to be defined.

render(): The function “render()” must be provided. It is used to show a preview of the element with its current settings.

xml(): Also the “xml()” function must be provided. This function is used to provide information for the XML export.

5. Results and Conclusion

5.1. Results

The display firmware is fully developed and ready to use. Also all goals for the software were reached. During the development process some problems appeared.

To simplify further development some of the problems are described in the following:

- The EVE core is a development from 4D Systems. It is still in a kind of “beta” state. Therefore, the platform has the following weaknesses:
 - The 4DGL language is very primitive and has only one type of variables.
 - The documentation is incomplete. Therefore, it is required to do practical tests to evaluate important specifications. This costs a lot of development time and makes it hard to determine the feasibility.
 - There are also some bugs in the compiler for the 4DGL language. For example it is not recommended to use “switch/case” statements, because it messes up the stack!
- There are also many bugs in the internal 4DGL functions. Some of them are listed here:
 - If you write a new file to the SDCard and the file has exact the length of “512” bytes, it is not possible to close the file handler.
 - It makes a difference using a fixed allocated global buffer or a dynamical allocated one. If you use a dynamical allocated buffer to fill a file with information, the file will be filled with blanks.
 - The string operations are bug-ridden. It needs a lot of time to get them to work how they should.
- The limited resources were also a big problem. Over 80% of the FLASH memory is used for the main program and nearly the whole SRAM is reserved. So during the development process often the space optimization was preferred. Also many parts of the program are exported to the SD-Card and gets loaded on demand, which reduce the over all performance.

The result is that a lot of code is included to the software to fix this bugs. Therefore, it is possible that a new revision of the display platform can cause instability of the program.

The development of the designer tool worked like expected. Java is a stable cross-platform-compatible language with lot of features. Therefore, a lot of development time was saved.

5.2. Conclusion

The GUI design has a long history. The expectations of the users are very different. In this thesis the used GUI elements are graphically orientated on “Windows 98”. Therefore, the elements are very simple and nearly every user knows how to interact with them.

The limited resources of the chosen platform are a big disadvantage. It would have been a better solution to use an extra processor board. For example Microchip provides a ready to use graphical display library [Mic].

On the other hand 4D systems announced a new graphic processor, which also includes the EVE core. Therefore, it is possible to optimize the firmware.

All together the result of this thesis is very usable for prototyping and limited-lot productions.

Bibliography

- [Lab12a] 4D Labs. *4DGL Programmers Reference Manual and Language Specifications*. 4D Systems, <http://www.4dsystems.com>, March 2012. Available at <http://www.4d-labs.com/downloads/4DGL-Docs/4DGL-Programmers-Reference-Manual-rev4.pdf>.
- [Lab12b] 4D Labs. *PICASO-GFX2 Embedded 4DGL Graphics Controller Datasheet*. 4D Systems, <http://www.4dsystems.com>, February 2012. Available at <http://www.4dsystems.com.au/downloads/Semiconductors/PICASO-GFX2/Docs/PICASO-GFX2-DS-rev4.pdf>.
- [Lab12c] 4D Labs. *PICASO-GFX2 Internal 4DGL Functions*. 4D Systems, <http://www.4dsystems.com>, June 2012. Available at <http://www.4dsystems.com.au/downloads/Semiconductors/PICASO-GFX2/Docs/PICASO-GFX2-4DGL-Internal-Functions-rev5.pdf>.
- [Mic] Microchip. Graphics displays. www.microchip.com/graphics/. [Online; accessed 01-August-2012].
- [Oraa] Oracle. How to install java. http://www.java.com/en/download/help/windows_manual_download.xml. [Online; accessed 01-August-2012].
- [Orab] Oracle. Java internationalization: Localization with resource-bundles. <http://java.sun.com/developer/technicalArticles/Intl/ResourceBundles/>. [Online; accessed 01-August-2012].
- [Sys] 4D System. Homepage. <http://www.4dsystems.com.au>. [Online; accessed 01-August-2012].

A. User Guide

This user guide is about how to design and set up the GUI designer firmware for the 4D System graphic display.

A.1. Preparation

At first some preparations have to be done to use the GUI:

Prepare the display: To use the 4D System graphic display the firmware and the program have to be loaded on the flash memory. This process has to be done once for each display. For more information read the Section A.1.1.

Install Java: You have to install Java to use the designer tool on your computer. The minimum requirement is version 6.0. For more information visit the Java homepage [Oraa].

Prepare micro SD-Card: The micro SD-Card has to be prepared. For more information read the Section A.1.2.

A.1.1. Display Preparation

It is not that simple to prepare the display. Read the following instruction carefully.

Requirements

The programming environment must be installed to prepare the 4D System display. Therefore, your computer needs the following system requirements:

- A Windows Operating System XP or higher
- A free USB port at least USB 1.1

Also the programming cable for the display is required. For more information about the programming cable visit the 4D System homepage [Sys].

Install 4D Workshop3

Download the setup for the “4D Workshop3” ¹ from the 4D System homepage [Sys]. Open it and follow the instructions.

Connect the display

Now you can connect the display over the programming cable with your computer. Therefore, ensure that a driver was found and installed correctly. When using windows XP you possibly will have issues.

Flash Firmware

You have to download the firmware from the 4D System homepage [Sys] first to flash it. You will find the file on the product page. The firmware revision **r30** is recommended. Newer firmwares may also work but there could be compatibility problems.

You must open the “4D Workshop” to flash the firmware on the display. When the program is open go to “Tools” and click on “PmmC Loader”.

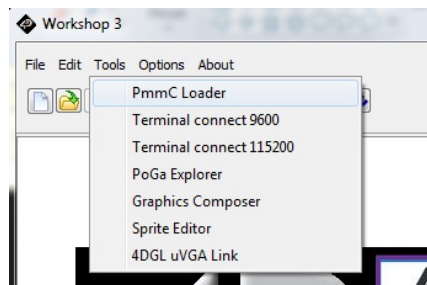


Figure A.1.: Open “PmmC Loader”

Then a new window gets open. Select the downloaded firmware and click on “Load”. The program should look like Figure A.2.

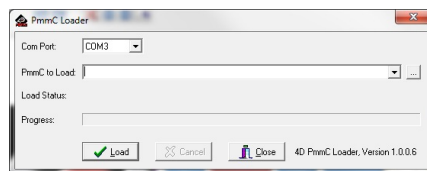


Figure A.2.: PmmC Loader

If the download is not working try to choose an other “Com Port” .

¹Newer Versions should also work!

Flash Program

Go back to the “4D Workshop” and open the file “main.4dg”. Figure A.3 shows how to program the display:

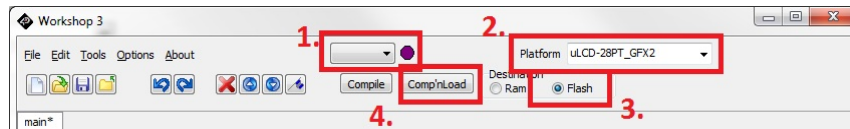


Figure A.3.: Load a program

1. Choose a comport. When the sign on the right side is blue you have chosen the right port.
2. Select the “Platform” of your display. The name can be found on the back side of the used graphic display.
3. Select “Flash” as the destination for the program. This is very important. Otherwise you will get an out of memory error while using the display.
4. Click the “Comp’nLoad” button to start the download.

The 4D System graphical display is now ready for usage.

A.1.2. Micro SD-Card Preparation

To use the micro SD-Card it must be formatted as “FAT16” . The following instructions only works on Windows computes.

Connect the micro SD-Card to your computer. Therefore, you can use a card reader. When the card is connected open “MY Computer”. Then choose the SD-Card devise and make a right click on it. Select the menu point “Format...”.

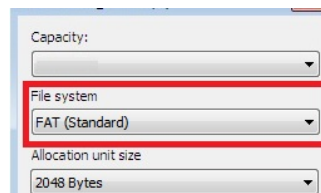


Figure A.4.: Format SD-Card

Now a new window should open. Select the same configurations as shown in Figure A.4. To start the formatting click on “Start”.

WARNING: All data on the SD-Card will be deleted!

A.2. Start Designer

The Designer is a Java program. Therefore, install Java first. For more informations read the Section A.1.

To start the designer tool just click two times on the “Designer.jar” file.

If nothing happens there may be an other solution. Open a shell. When using windows press “STRG + R ”. The window shown in Figure A.5 will appear.

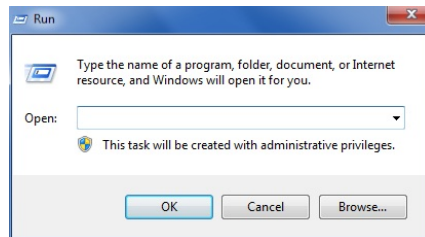


Figure A.5.: Open designer tool

Click on the “Browse...” button and select the “Designer.jar” file. Next add “java -jar ” on the beginning of the text box. Do not forget to add a space between the added text and the file name. To start the designer click on “OK”.

A.3. Menu

This section is about the menu of the designer shown in Figure A.6.

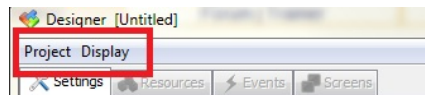


Figure A.6.: Menu of the designer

A.3.1. Project

The following option are available in the “Project” submenu:

New: Click on “New” item to crate a new project. The old unsaved data will be discarded.

Load: Here you can load a saved project.

Save: Here you can save the changings of the open project. If the project is new you have to chose a file name for saving.

Save As...: If you want to save the open project under a new name, click on “Save As...” item.

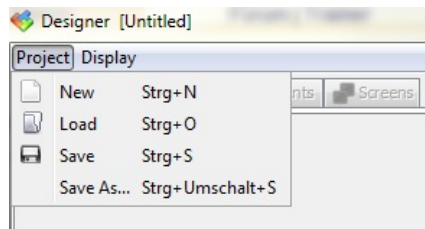


Figure A.7.: Project submenu of the designer

A.3.2. Export

If you have finished the design or if you want to test it, you have to use the export function. Click on the "Export" submenu item shown in Figure A.8.



Figure A.8.: Submenu item export

Therefore, a new window appears shown in Figure A.9.

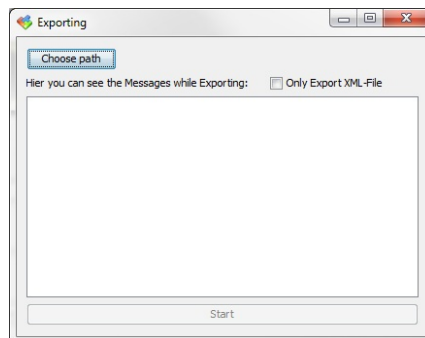


Figure A.9.: Export window

The following steps have to be made to export the data to the micro SD-Card:

1. Click on "Choose path" to open the root folder of the micro SD-Card. Therefore, connect and prepare the card to the computer. For more information read the Section A.1.2.
2. If you have made only little changes, you can activate the option "Only Export XML-File". Then no images or system data gets exported. This will speed up the export process, but it only works if you have made only minor changes after having made a full export on the same card before.

3. Click on the “Start” button.

WARNING: All files in the selected folder get removed.

There may occur errors during the export. You can click on the messages to get to the right tab in the designer tool to solve the problems. If there are red messages, they mean that there are errors and the export was not successful. All other messages are hints and may not cause problems.

A.4. Tab Settings

This section is about the settings of a project.

A.4.1. Basic Setting

You have to set the basic settings before you can start designing.

WARNING: The basic settings are irrevocable. If you want to change them you have to start a new project.

Figure A.10 shows the basic settings.

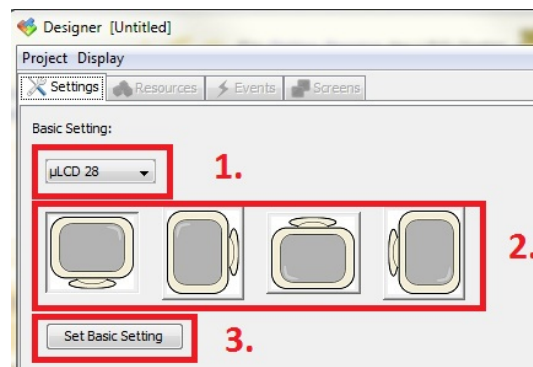


Figure A.10.: Basic settings overview

The following settings have to be applied:

1. Here you have to choose the platform. You can find the name of the platform on the back side of the display.
2. Next you have to choose the orientation of the display. When the basic settings are locked, you can still rotate the orientation by 180 degree.
3. Click “Set Basic Settings” to lock the settings.

A.4.2. Other Settings

All other settings can always be changed. The following list describes the settings on the settings tab:

Serial port configuration: Here you can set the communication speed between the display and the application. For more information read the Section A.10.

Brightness: Some platforms have the possibility to change the brightness. When the track bar is enabled you can manipulate the brightness of the display.

Start Event: Here you can select the event which is triggered when the display is ready after the start-up. For more information about the events read the Section A.6.

Start Screen: You have to choose a start screen. On start-up the selected screen will be shown on the display. For more information about the screens read Section A.7.

A.5. Tab Resources

Resources are like variables. They can be changed during the runtime time. There are many possibilities to change resources. One possibility is to change them over the application interface. For more information read the Section A.10.

So resources hold a changeable information. This information can be displayed in many ways or read out over the application interface. Therefore, resources are the main element to manipulate the GUI and show information to the user.

A.5.1. Resource Types

There are different types of resources:

String: A string can hold any kind of text. Also the length of the string is user-defined.

Boolean: Can only hold two states. One is “true” and the other one is “false”.

Color: Holds a color. The color is represented as 3 digit hex number. The color is coded in RGB mode and each digit is for one elementary color.

Number: A Number has 4 significant digits, but the position of these digits is fixed. There are 7 different types of numbers. The difference between the types is the number of the significant and therefore the rendering. For example “Number 1” has 1 significant in the integer part and 3 in the fractional part.

The “Number 2” has 2 significant in the integer part and 2 in the fractional part.

The last type is “Number 7”, which has 4 significant followed by 3 zeros and no fractional part.

A.5.2. Manage Resources

Figure A.11 shows the tab of the designer to manage the resources.

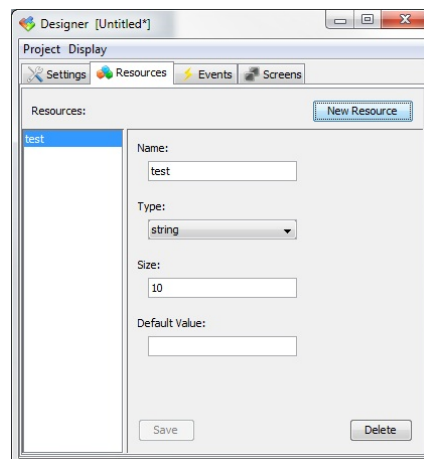


Figure A.11.: Resources overview

Add Resources: To add a resource just click on the “New Resource” button. You have to choose a name for the resource. For more information about the name of a resource read the Section A.5.3.

Remove Resources: If you want to delete a resource, you have to select it and press the “Delete” button. If the resource is used somewhere you will get a warning, but the deletion is still possible.

Edit Resources: First select the resource you want to edit. It is also possible to select multiple resource and edit them together. After you have changed the settings click on the “Save” button, otherwise the changings will get lost.

A.5.3. Resource Settings

A resource has different settings as shown in Figure A.11:

Name: The name of the resource is very important. You need it for the application interface. Read more about the interface in Section A.10. The name is also used to choose the resource for events or for the screen elements.

Type: Here you can choose the type of a resource. For more information about the types read Section A.5.1.

Size: If the resource type has a user-defined length, it is possible to set the length here. You have to keep in mind that the display has limited memory resources.

Default Value: On start-up of the display the resource gets initialised with the default value. You can choose any value, but it must be valid for the type of resource.

A.6. Tab Events

Events are used to inform the application over the application interface that something happened. For example the click on a button can trigger an event. For more information about the transmission of events read Section A.10.

A.6.1. Manage Events

Figure A.12 shows the tab of the designer to manage the events.

Add Event: To add an event just click on the “New Event” button. The name is for the user of the designer and will not get exported to the display.

Remove Event: If you want to delete an event, you have to select it and press the “Delete” button. If the event is used somewhere you will get warning, but the deletion is still possible.

Edit Resources: First select the event you want to edit. It is also possible to select multiple events and edit them together. After you have changed the settings click on the “Save” button, otherwise the changings will get lost.

A.6.2. Event Settings

An Event is based on two settings:

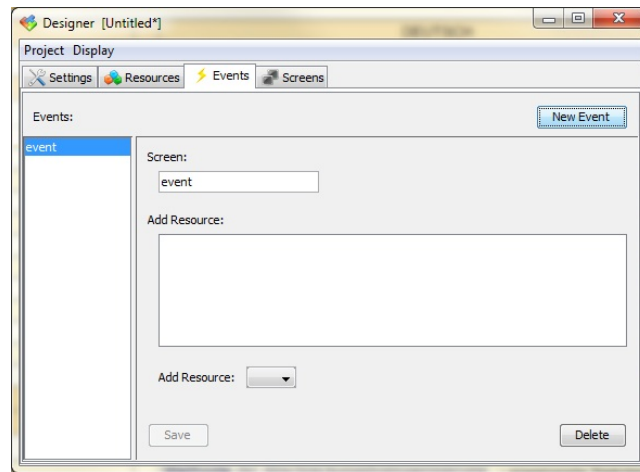


Figure A.12.: Events overview

Name: The name of the resource is only used in the designer and will not be exported to the graphic display. It is just a help for the user to choose the right event in a list.

Text: Here you can specify the text which will be send to the application when an event is triggered. For more information about the application interface read Section A.10.

The text can also contain resources. When an event is triggered the resources get replaced with there current value. To add a resource to the text just choose it from the “Add Resource” combo box.

A.7. Tab Screens

The screens are the main part of the designer. You can create multiple screens, but you can only edit one screen at the same time.

A.7.1. Manage Screens

Figure A.13 shows the screen tab.

Add Screen: If you want to add a screen, just click on the “New Screen” button. Then you have to choose a name for the screen. The name is only for internal use and will not be exported to the display.

Remove Screen: Click on the “Remove Screen” button to remove a screen.

Screen Settings: To manipulate the settings of a screen click on the “Screen settings” button. For more information about the screen settings read Section A.7.2.

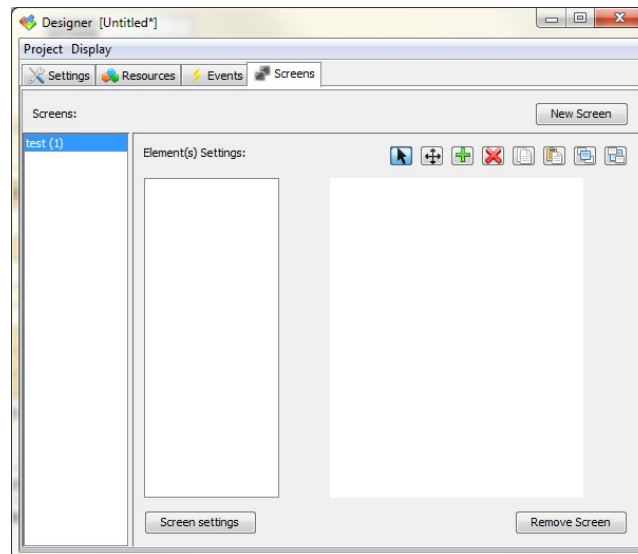


Figure A.13.: Screens overview

A.7.2. Screens Settings

Figure A.14 shows the settings window for a screen.

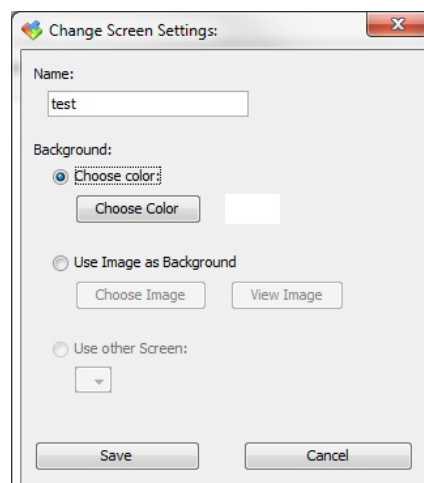


Figure A.14.: Screen settings overview

Name: The name of a screen is only for internal use. It will not be exported to the display. It is possible to switch between screens over the application interface. Therefore, in the screen list, shown in Figure A.13, each screen has a number. The number is used to switch to a screen over the interface. For more information read the Section A.10.

Background: Here you can select a background for the screen:

Color: When “Choose color” is selected, the background is filled with the specified color.

Image: You can select “Use Image as Background” to use an image for the background. Press the “Choose Image” button, to select an image. Therefore, the “Scale Image” window shows up. For more information about scaling images read Section A.9.

Screen: It is possible to use an other screen for the background. Therefore, all elements from that screen will also be rendered on this screen. The graphic display itself does not support parent screens. Therefore, during the export all elements from the parent screen get copied to this screen. So this function is only a tool to reduce design time. For example if you have a menu on multiple screens, you can design a menu screen and use it for all screens where a menu is planned. In combination with a tab page element this is a good approach.

A.7.3. Designer Panel

Figure A.15 shows the design panel. The panel is used to manage elements on the screen.

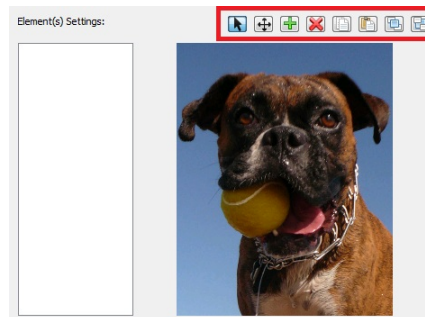



Figure A.15.: Screen designer panel

The following functions are provided:

Select : This function is normally selected. With this tool you can select one or multiple elements. If you click on an element it gets selected, but the last selection get lost.


If you press “Shift” while clicking the element, the element selection gets inverted. Therefore, a selected element will get removed from the selection and an unselected element is added to the selection.

It is also possible to select elements in an area. If you click and drag the mouse all elements in the defined area get selected, but the last selection


gets lost. If you press “Shift” during this action the selected elements get added to the last selection.

If you click and drag a selected element, all selected elements are moved relative to the mouse position.

The properties of the selected elements can be changed. When an element is selected the available properties are shown in the list on the left side. If you click on one property, a window will popup to change it. It is also possible to change properties of multiple elements.

Scale : Select an element first before using this function. To use this function click on the function icon. It provides the possibility to change the size of a selected element. If you want to change the size, you only have to drag and drop the dashed line. It is also possible to resize multiple elements.

Furthermore you can move the selected elements. Therefore, click on a selected element and drag it.

Add : This function is used to add an element to the screen. If you want to do so, click on the icon. Then you have to define an area by click and drag over the screen. Figure A.16 shows the windows, which pop-up after have defined an area. Select a type for the element you want to add. For

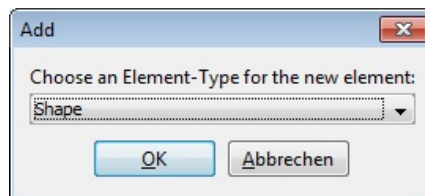





Figure A.16.: Add an element to the screen


more information about the element type read Section A.8.

Delete : If you click on this function icon, all selected elements will be removed from the screen.

Copy : It is possible to copy selected elements. If you want to copy an element, just select it and click on the function icon. It is also possible to copy multiple elements.

Past : If you have copied some elements, you can paste them by clicking on the function icon.

Bring to Front : When one element overlaps another, one is in the front and the other one will be overlapped. With this function it is possible to bring the element which is overlapped to the front.

Bring to Back : This function acts like “Bring to Front”, but it will bring the element from the front to the back.

A.8. Screen Elements

You can place different sorts of elements on a screen. In this section the properties of these elements are described:

A.8.1. General Properties

There are properties, which are shared with multiple element types:

Mouse over Event You can select an event, which gets fired when the user presses on this element. The event is fired before the user releases the click.

Mouse out Event Here you can select an event, which is fired when the user starts pressing over the element, but releases the click outside of the element.

Mouse click Event This property has the same behavior than “Mouse out Event”, but the event gets fired when the click is released over the same element.

All events are fired before the element logic gets the information. So if an element changes a resource on a click event, the resource will not be changed in the event text of this event. So there are element specific events, which are fired after the resource gets changed.

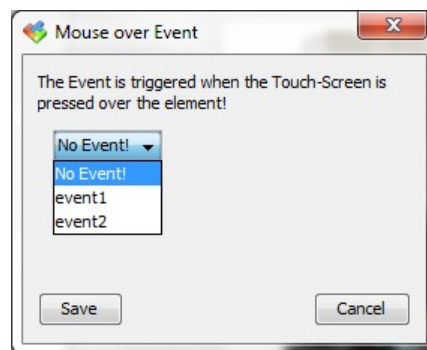


Figure A.17.: Select an event

For more information about creating an event read the Section A.6. Figure A.17 shows the property window to select an event.

Color: This property defines the main color of an element. For “Shape” element, it defines the color of the painted figure. For other elements its the color of the text.

Back-Color: If you want to change the background color of an element, this property is used.

Border-Color: The most elements have a boarder. Therefore, this property is used to change the boarder color.

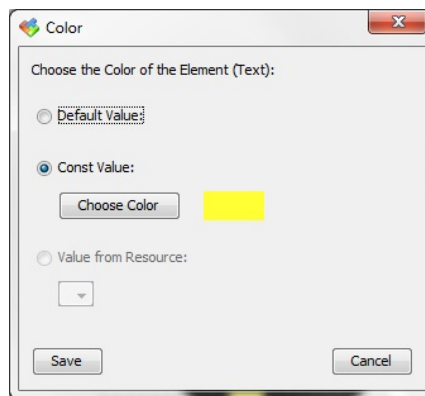


Figure A.18.: Select a color

For all color properties you can choose between a fixed color or a color of a resource. Figure A.18 shows the color property window.

Text: Many elements have the possibility to show text. This property enables you to specify the text. Figure A.19 shows the property window.

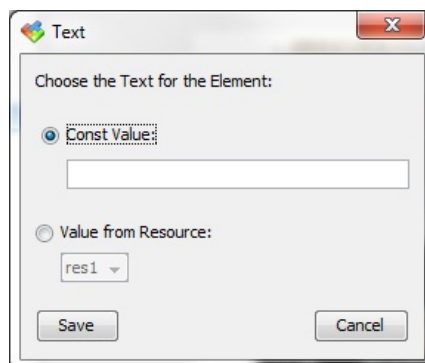


Figure A.19.: Choose a Text

You can enter a fixed text or it is possible to choose a resource. When you choose a resource the text of the resource is shown.

Font: Here it is possible to change the font of the text. The “System” font is integrated in the firmware of the display. Therefore, it is much faster and

uses less memory than the others. If there is not enough memory left to load a font, the “System” font is used instead. Figure A.20 shows the property window.



Figure A.20.: Choose a font

A.8.2. Shape

The “Shape” element is used to display a geometric figure. The following properties are not general:

Filled: Here you can choose if the shape should be filled or not. It is also possible to use a boolean resource.

Shape: This property window is a bit more complex, than the others. Figure A.21 shows the window.

If you want to choose a shape, following steps have to be done:

1. Select the type of the shape.
2. If you want to make the shape bigger² than the element you can change the zoom.
3. Select the point you want to change.
4. Click on the position the selected point should be set.

It is only possible to resize the element when no shape is selected.

²The overlaying shape will not be rendered.

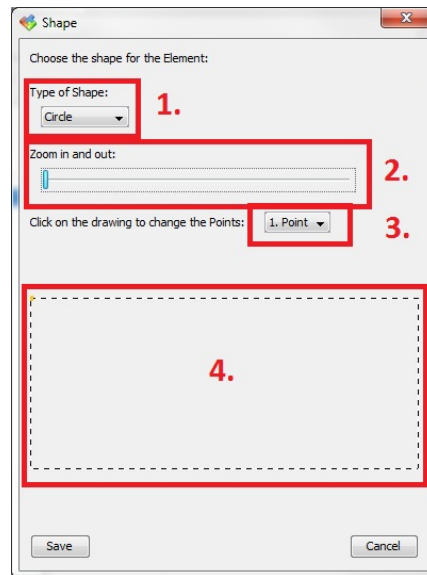


Figure A.21.: Choose a shape

A.8.3. Image

The image element is used to display images. The most important property is the “Image” property. There are two modes:

Easy Mode: This mode is recommended. Figure A.22 shows the part of the property window. When you check “show” or “over” the image scaler opens. For more information read Section A.9.

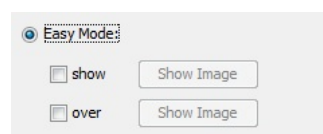


Figure A.22.: Image property easy mode

It is only necessary to select an image for the “show” option. The “over” option is used to show a different image while the user clicks on the image element. If no second image is selected, there will be no change during a click on the element.

If any option is selected the image element is not scalable.

Expert Mode: The “Expert” mode works the same way as the “Easy” mode, but it is possible to add more than two images and change them over a resource. Figure A.23 shows the second part of the property window.

To add an image to the list click on the “+” button. First the image scaler gets open. For more information read Section A.9. Then you have

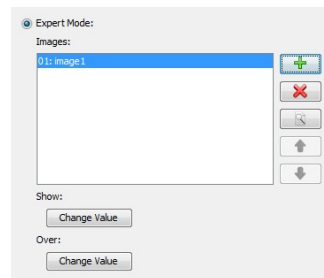


Figure A.23.: Image property expert mode

to enter a name for the images. The name is only for internal usage and will not be exported to the display.

The “X” button is used to remove an selected image from the list.

Each image in the list gets an unique number. This number is used to select the corresponding image.

Now you can choose a fix number or a resource³ for “Show” and “Over”. The specified images from “Over” is shown while the user clicks on the element. Otherwise the image specified by “Show” is shown.

If any image is in the list the element is not scalable.

A.8.4. Text

The text element is used to show text. Therefore, the following property can be set:

Text Changed Event: This event will be fired, if the text of the resource chosen in the “Text-Resource” property has been changed.

Multiline: Indicates that line breaks will be added to the text automatically. Otherwise the text will be rendered in one line. It is not possible to add line brakes to a text manually.

Scroll: Enables the possibility to scroll the text. Therefore, the user is able to scroll the text by clicking on the text element.

Textbox: This property is used to change the style of the element. When this setting is activated the text element looks like a textbox.

Autoscroll: If this property is enabled the text of the element will be scrolled automatically.

Center: It is possible to center the text. To center the text just activate this property.

³The resource type have to be “Number 4”

Text-Resource: If the user clicks on the text box, it is possible to open an “on screen editor”. If a resource is chosen for this property, it will be edited when the user clicks on the element.

A.8.5. Button

A button has the following extra properties:

Button Clicked Event This event is fired when the button was clicked and the button “Resource” property action is handled. The event is also fired when no action is defined.

Resource Here it is possible to define an action, which will be executed when the button is pressed. Figure A.24 shows the property window.

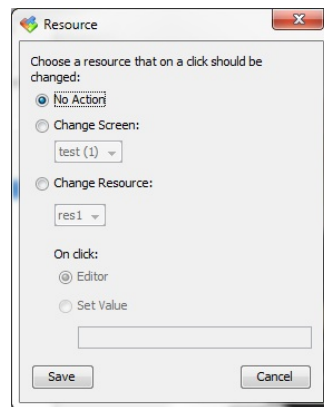


Figure A.24.: Button resource property

The following actions are selectable:

No Action: On click nothing happens. Only the “Button Clicked Event” is fired.

Change Screen: When this is selected, it is possible so choose a screen. If the user clicks on the button the screen will be changed.

Change Resource: Here it is possible to manipulate a resource. If “Editor” is selected, the user will get to a text editor when the button is pressed. On “Set Value” the resource is changed to the given value.

A.8.6. Checkbox

The checkbox element is also a radiobox element. The following settings are not general:

(Un)Checked Event: This event is fired on a click and after a resource change, if one is defined.

Radiobox: Here you can define the style of the element. If the property is set, the element looks like a radio box.

Value: This property is to set the state of box. It can be set to a constant value or to a resource. Therefore, all resource types are possible. You only have to specify a value. When the resource is equal to it the box will be checked.

Resource: It is possible to change a resource, when the user clicks on the element. You can define what will happen when the box is unchecked or checked. The property window is shown in Figure A.25.

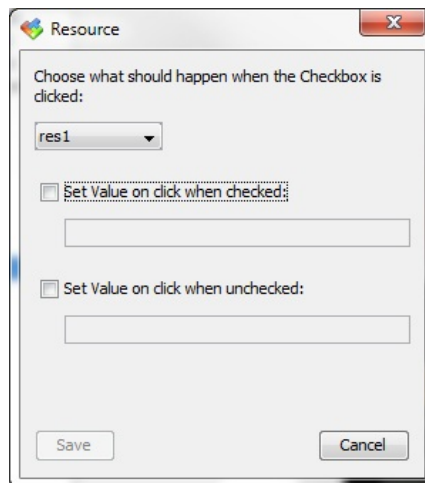


Figure A.25.: Checkbox resource property

A.8.7. Progressbar

The progressbar is a very simple element to show the progress of a process. There is only one property, which is not used generally:

Value: Here you can choose a resource⁴ When no resource is selected or the value of the resource is not between “0” and “100”, the “Marquee Style” is enabled. In this mode the rendered bar is moving, but it does not show what proportion of the progress is complete. Otherwise the value of the resource is shown.

⁴The resource type have to be “Number 4”.

A.8.8. Trackbar

The trackbar can be used to change a “Number” resource. The following options are not used generally:

Value Changed Event: This event will be fired after the corresponding resource is changed.

Length: Here it is possible to change the length of the bar.

Value: Figure A.26 shows the property window.

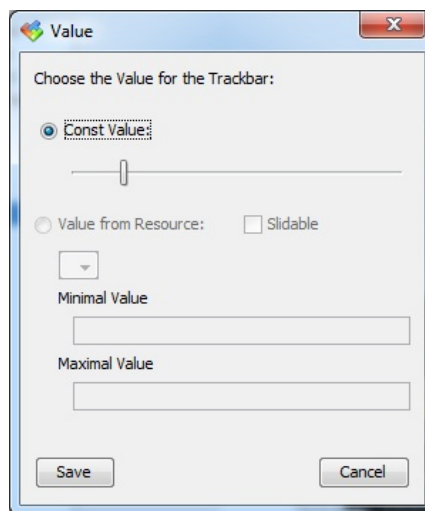


Figure A.26.: Trackbar value property

If you choose “Const Value”, you can directly pick the position of the bar, but it is fixed during runtime.

It is also possible to choose a resource for the position. All number types are allowed. If you activate “Slidable”, it is possible for a user to change the resource during the runtime.

Realtime: If you have activated “Slidable”, this property defines the behaviour during the sliding. When “Realtime” is activated, the resource gets changed during the drag of the bar. Therefore, the “Value Changed Event” is fired continuously. In normal mode the resource gets changed and the event is fired when the user releases the bar after sliding.

A.8.9. Tabpage

The Tabpage is used to change the screen easily. It shows the screen names and when the user clicks on one, the corresponding screen is shown. Therefore,

tabpages are very useful in combination with a “parent screen”. For more information read Section A.7.2. The following properties are not used general:

Screen Changing Event: This event is triggered when the user changes the screen.

WARNING: When you use the screen resource in the event text, it will not be up to date.

Screens: Figure A.27 shows the property window.

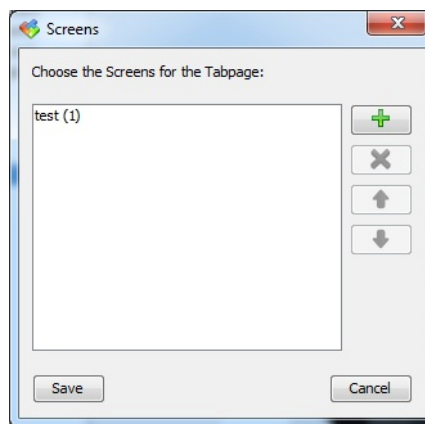


Figure A.27.: Tabpage screens property

Here you can choose the screens for the tabpage. If you want to change the name of a screen, you have to switch to the screen and click on “Screen settings”.

A.9. Image Scaler

On some places you have to choose an image. Therefore, the “Scale Image” window shows up. The window is shown in Figure A.28.

1. Click on the “Open” button to choose an images. The image is then shown in the middle of the window.
2. The box represents the selected area of the image. You can move the box by clicking in the middle and dragging it. It is also possible to change the size of the box. When the mouse is over the dashed line you can click and drag it. Therefore, the size of the box is changing.
3. The two icons are very helpful. The icon on the left side is used to rotate the images. The icon on the right side helps with the scaling of the box. When the icon is activated the width-to-height ratio of the scale box is fixed. Therefore, the scaled image will not be distorted.

If the image in the box is the way you want it, click on the “Use Image” button.



Figure A.28.: Scale image window

A.10. Application Interface

In this section the interface between the display and the application is described. The interface is based on a “RS-232” port. That board uses TTL levels. For more information about the port read [Lab12b, Section 2.3].

There are three different functions implemented:

Events: Is used to notify the application that something happened. For more information about creating events read Section A.6.

Read Resource: This function enables the application to read a resource. For more information about resources read Section A.5.

Write Resource: This function enables the application to change a resource. For more information about resources read Section A.5.

The communication protocol is based on bytes. Events can be triggered asynchronously. Therefore, you have to consider that events can be triggered during a read or write operation.

The protocol differentiates between character bytes and control bytes. A character byte is in the range of “32” (0x20) to “126” (0x7E). Therefore, only spaces and normal characters are included. All other bytes are control bytes.

A.10.1. Events

An event is asynchronously sent to the application. It does not matter in which state the interface is. You have to consider this in your application.

An event is processed the following way:

Start: To identify an event the control byte with the value “1” (0x01) is sent.

Text: The text of the event is send after the control byte. For more information about creating events read Section A.6.

End: To indicate the end of the event text the control byte with the value “10” (0x0A) is send.

The following table shows an example for an event with the text “Event1”:

Received from the screen:								
Value	0x01	0x45	0x76	0x65	0x6E	0x74	0x31	0x0A
Text		E	v	e	n	t	1	↵

The event handling has the highest priority. It is not possible that this function gets interrupted by an other interface function. Therefore, it is not necessary to look for other control characters during an event receiving.

A.10.2. Read Resource

To read the content of a resource, this function is used. For more information about resources read Section A.5. The following steps are necessary:

Start Symbol: You have to send the start symbol to sync the interface. The start symbol is “:” or “58” (0xA3).

Name: Now the name of the resource has to be send. One byte is send for each character.

Mode Symbol: The mode symbol is used to indicate a reading access. The mode symbol is “?” or “63” (0x3F).

End: To request an answer the control byte with the value “10” (0x0A) is send.

There will be an answer from the display. You have to consider that an event can interrupt the communication. There are two possible answers:

Error: If something went wrong, the error control byte is send. The error control has the value “0” (0x00).

Value: If all went right, the control byte with the value “2” (0x02) is send. Then the value of the requested resource is following and at the end the control byte with the value “10” (0x0A) is send.

The following table shows an example for reading a resource called ”Test”:

Send to the screen:							
Value	0xA3	0x54	0x65	0x73	0x74	0x3F	0x0A
Text	:	T	e	s	t	?	↵
Received from the screen:							
Value	0x02	0x54	0x65	0x78	0x74	0x0A	
Text		T	e	x	t	↵	

A.10.3. Write Resource

To write the content of a resource, this function is used. For more information about resources read section A.5. The following steps are necessary:

Start Symbol: You have to send the start symbol to sync the interface. The start symbol is “:” or “58” (0xA3).

Name: Now the name of the resource has to be send. One byte is send for each character.

Mode Symbol: The mode symbol is used to indicate a reading access. The mode symbol is “=” or “61” (0x3D).

Value: Now the new content of the resource is send. There are only character bytes allowed.

End: To request an answer the control byte with the value “10” (0x0A) is send.

There will be an answer from the display. You have to consider that an event can interrupt the communication. There are two possible answers:

Error: If something went wrong, the error control byte is send. The error control has the value 0 (0x00).

Resource changed: If all went the way expected, the control byte with the value “1” (0x01) is send.

The following table shows an example for writing ”ABC” to a resource called ”Test”:

Send to the screen:										
Value	0xA3	0x54	0x65	0x73	0x74	0x3D	0x41	0x42	0x43	0x0A
Text	:	T	e	s	t	=	A	B	C	↵
Received from the screen:										
Value	0x01									
Text										

A.10.4. Special Resources

There are some reserved resource with special purpose:

screen: The resource with the name “screen” can be used to poll or set the active screen.

render: Here it is possible to disable the rendering. If you set the resource with the name “render” to “false”, the touch is disabled and the display is not updated. Therefore, also no events can happen during rendering is disabled. The speed of the interface will increase. To resume to the normal mode, just set the “render” resource back to “true”.

B. Display Design File

Here the XML file is described, which defines the whole GUI and the settings of the display. This file, which is located in the root folder of the SD-Card, is named “display.xml”.

The XML file has the following structure:

There is only one root element with the tag name “display”. No comments are allowed and the attribute definition must use double quotes. A double quote in a value can be escaped by a backslash. Furthermore there is no text in element bodies allowed, only line breaks are allowed.

Element Name: “display”

Attributes:

- **comspeed:** The value describes the tenth of the communication speed. Default value is “11520” for 115200 baud rate.
- **mode:** This attribute defines the orientation of the display. Allowed values are: “0”, “90”, “180”, “170”. Default value is “0”.
- **contrast:** It defines the contrast of the display. Allowed range is from “1” to “9”. Default value is “9”.

Body: This element must have a body with three elements. The names of the elements must be “resources”, “events” and “screens”. No other elements are allowed.

Element Name: “resources”

Attributes: No Attributes.

Body: This element must have a body even if it has no children. The only allowed element name for a children is “resource”.

Element Name: “resource”

Attributes:

- **name:** Here you have to specify the name of the resource. The names “screen” and “render” are reserved.
- **type:** Also the type of the resource has to be defined. The following types are allowed: “string”, “bool”, “color”, “number1”, “number2”, “number3”, “number4”, “number5”, “number6”, “number7”.

- **size:** Here you can define the size of the memory for the resource in bytes.
- **default:** The default value for the resource must also be defined.

For more information read Section A.5.

Body: No body is allowed.

Element Name: “events”

Attributes:

- **startevent:** The event is triggered, when the display is stated.

Body: This element must have a body, even if it has no children. The only allowed element name for a child is “event”

Element Name: “event”

Attributes:

- **id:** ID of the event must be defined. The allowed range for the ID is “1” to “255”.
- **sendXX:** The symbols “XX” have to be replaced with a number starting from “1” to a maximum of “99”. You can fill this attributes with text or resources values. To use a resource as text just use the following attribute value: “RES:XXXX” and replace “XXXX” with the name of the resource you want to use.

For more information read Section A.5.

Body: No body is allowed.

Element Name: “screens”

Attributes:

- **startscreen:** The screen is shown, when the display is stated.

Body: This element must have a body and at least one child in it. The only allowed element name for a children is “screen”

Element Name: “screen”

Attributes:

- **id:** You must define an ID for the screen. The allowed range for the ID is “1” to “255”.
- **image:** It is possible to define a background image. Therefore, use the image file name for the attribute value.

- **color:** Define a background color if no images is defined. A three digit hex number is expected.

For more information read Section A.7.2.

Body: This element must have a body even if it has no children. The only allowed element name for a children is “element”

Element Name: “element”

Attributes: For all attributes of this element, it is possible to use a resource instate of the fixed text. Just use “RES:XXX” as value for an attribute and replace “XXX” with the resource name.

The following attributes are the same for all element types:

- **type:** Here you have to specify the type of the element.
- **x, y, width, height:** These attributes are used to define the position and size of an element. It is not allowed to use resources.
- **touchover:** Here you can specify an event number. This event is fired, when the user starts pressing on the element.
- **touchout:** Here you can specify an event number. This event is fired, when the user starts pressing on the element and stops out of it.
- **touchclick:** Here you can specify an event number. This event is fired, when the user clicks on the element.

For element type specific attributes read the comments in the source code. For more information about the element types read Section A.8.

Body: No body is allowed.