EMBIEN TECHNOLOGIES

Sparklet – Embedded GUI Library User Manual

No 3, Sankarapandian Street,
Madurai, India 625017
www.embien.com

Table of Contents

Table of Tables

Table of Figures

Revision History

Revision	Author	Description
1.0 – 29 Jul 2015	SP	Initial release

Chapter 1

Introduction

This chapter provides a brief introduction about the Sparklet Embedded GUI library and sets the context for the further reading of the document,

1.1 Purpose of the Document

Sparklet Embedded UI library is a light weight library created to run on top of resource critical embedded systems. Intended for a wide range of applications not limited to Industrial HMI's, Medical devices, consumer electronics, auto infotainment panels, Sparklet provides a very rich User Experience that can be customized to the tastes of the vendor and the industry.

This document provides a complete overview of development for Sparklet embedded GUI library including the tools available for development, procedures for simulation in host PC before running in the actual target etc.

1.2 Scope of the Document

This document explains the internals of the Sparklet library, including its architecture, components that make it up. Also the document explains various tools and techniques associated with the Sparklet development.

This document provides both the user perspective - necessary for developing applications on top of Sparklet and the developer perspective - needed for customizing and improving functionalities of the library.

Details of the Application Programmer Interface i.e. APIs that are available to the user are available as a separate document called Sparklet API Manual in http:\\www.embien.com\downloads page.

Information about using Flint Plug-in to develop screens for Sparklet is not documented here. Instead it is available as a separate document in http:\\www.embien.com\downloads page.

1.3 Conventions Used

The below table gives a summary of conventions used throughout the document.

int main ()	Source code, most likely in C language
init_sgui	Function available in Sparklet UI library

Table 1 Conventions used

Chapter 2

Beginning Development

Reader is introduced to various tools associated with the Sparklet UI development including the Flint IDE, RAPIDSEA etc.

2.1 Overview

Sparklet Embedded UI library is a stand-alone library that can be used with any customer application. Generally it is compiled together with the customer logic. Though it is sufficient to develop screens, Embien offers more tools to aid firmware development for the user. These tools include

- Flint - Eclipse Plug-in - for Sparklet Screen development
- RAPIDSEA - Embien Application library for logic development

Details of these tools are explained below.

2.2 Sparklet GUI Library

Sparklet GUI library is provided in source form to the customers along with few examples. This highly optimized library is organized in a developer friendly fashion as explained in detail in the next chapter. .

2.3 Flint IDE

Developing screens and arranging them for proper layout is a tedious process as well as a time consuming one. Considering this, Embien has developed Flint - a GUI design tool. Flint is implemented as a plug-in to eclipse, taking advantage of the advanced features of the hugely successful platform. Installable across a range of Eclipse versions, this tool enables users to design the UI with a simple drag and drop approach. Further features and functionalities of each widget can be configured with convenience of a mouse, without even going in to the details of the UI implementation.

This tool along with the procedure to use it is detailed in a separate document ***Flint User Manual*** available from downloads page of our website.

2.4 RAPIDSEA Libraries

RAPIDSEA - RAPId Deployment Suite for Embedded Applications is a collection of embedded system functionalities that can be sued to develop and deploy embedded firmware. RAPIDSEA supports most popular architectures and targets including Arduino, FRDM boards, PCI MCUs, Raspberry PI etc. RAPIDSEA can be configured graphically using Flint IDE and compiled using standard compiler for the platform.

For more details on the same, kindly visit our website.

Chapter 3

Architecture

This chapter provides insight to the design principles of Sparklet, describes the internals of the UI library and about the different logical layers in which it is organized.

3.1 Design Principle

Before going in to the details of the Sparklet Embedded UI library, it is important to understand the principles behind the design of the same. Following considerations are factored during the design.

- Source Code in ANSI C - For use with standard Embedded systems
- Platform independent design
- Highly configurable
- Modular architecture
- Scalable design
- Support for multiple applications
- One App-One Window design
- Run with and without OS/RTOS
- Support for modern UI aesthetics
- Support for multi-touch
- Leverage underlying hardware functionalities as much as possible

Sparklet has successfully implemented most of these principles and, in each successive revisions, is moving forward towards a more universal and advanced GUI library.

With this, we will go in to the architecture of the library and follow it up with detailed discussion on the components.

3.2 Library Architecture

The below block diagram depicts the architecture of the Sparklet Embedded GUI library.

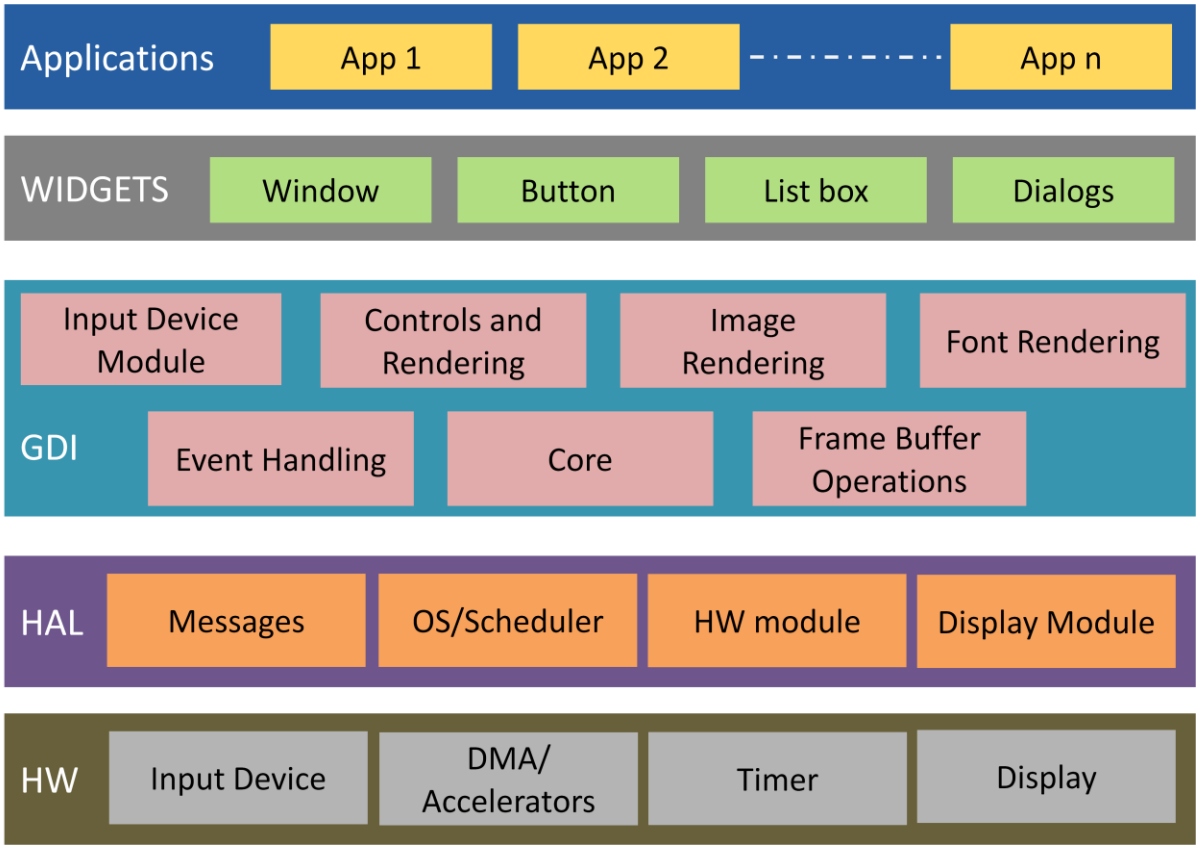

Figure 1 Block Diagram

Sparklet library is organized as 4 layers, each represented as a large horizontal rectangles sitting above the underlying hardware. They are

- Hardware Abstraction Layer - HAL
- Graphics Device Interface
- Widgets
- Applications

These horizontal layers are again made up on multiple components. Each of the small boxes represents a component in the library that performs a particular functionality. These components could be implemented either as a single file, across multiple source files or a part of a single file. These components have clearly defined entry points through which other components interact. Also they are provided with mechanisms such as callback to communicate asynchronously.

Though technically Hardware and Applications layer are not a part of Sparklet, they are nevertheless discussed in detail it is important to understand the overall functionality.

3.3 Hardware

Sparklet library is designed to run on a low end target. It could be run on a low power microcontroller that is clocked at a few tens of MHz Though faster the processor, it is better the performance, it is possible to get best results with careful tuning and configuration of the library and hardware.

Memory is also a very important criterion as the controls/widget to be rendered occupies memory. Most likely the screen to be painted is also to be stored in the memory in a section called frame buffer. Sparklet provides numerous configuration options that can be used to optimize the memory requirements of the library. These are discussed later. For now, we will look in to some of the major hardware components needed for the Sparklet library.

3.3.1 Display

Obviously Display is the first and foremost hardware component needed to run any UI library. There are different kinds of Display technologies available such as TFT LCD, STN LCD, LED, HDMI etc. These displays could be of any dimension i.e. resolution specified by width x height. They may have different color depths - represented by bpp (bits per pixel). Displays are usually interfaced with the microcontroller with display controllers that are necessary for generating the timing and for transferring data from the frame buffer memory to the display. Some types of displays may have frame buffers internally and are interfaced with the MCU over a general bus interface like SPI/I2C.

3.3.2 Input Devices

Input devices are needed to help user interact with the system. Different kinds of input devices are available such as touch controllers, mouse, joystick, keyboard etc. Nowadays touch panels are the most common type of input device being used. They are generally connected to the microcontroller using over interfaces like I2C or SPI along with dedicated lines for interrupt to notify the processor of touch events

3.3.3 Timer

Though timers are not directly needed for the Sparklet to function, there is a need for a mechanism to track time between events, invoking timeout handlers etc. Since timers are always available in the SoC being used, they are the best way to implement timing. Timer implementation is highly specific to the device being used.

3.3.4 Misc Components

There are other components that can be available in the hardware like the DMA engine, Bit-blit engines etc. These can be used to optimize the performance of the UI library DMA engines can be used to take copy of frame buffers, transfer chunk of data etc. Bit-blit engine can be used to transfer

rectangular regions as needed by the UI library. Hardware accelerators and GPUs can help performing dedicated draw operations faster.

3.4 HAL

As explained in the earlier section, there exist different kinds of devices on which the Sparklet UI library might be running. In keeping with the design principles, the Sparklet library is designed to run independent of them. This is achieved by a HAL – Hardware Abstraction Layer which needs to export certain functions to the top layer. This section explains the HAL layer along with the list of functions to be supported to enable Sparklet on the device.

3.4.1 Display

Whether the display uses frame buffer or not, the following functions are used by the library to do any draw operation on the display.

Function Name	Description
arch_draw_pixel	Draw a pixel at given position with given color
arch_draw_hor_line	Draw a horizontal line
arch_draw_ver_line	Draw a vertical line
arch_draw_filled_rect	Draw a filled rectangle

Table 2 HAL functions for display manipulation

As these form the basic operations, any complex shape can be formed with these. For detailed information of the same, kindly refer to the Sparklet API Manual.

It is important that these functions are implemented in the best way possible for the given hardware. For example, if there are options to any of these by hardware like GPU or 2D accelerators, it is recommended to follow the same, to achieve the best performance.

3.4.2 Input Device

Sparklet, as of now, understand only input events as mouse events. It respective of the input device technology, the data is to be given in using mouse events. The function,***sevt_post_mouse_event,*** must be used to provide the same to Sparklet. So even touch panel events, should be fed as

User Touch Action	Event to generate
User touches the panel	SEVT_MOUSE_LB_DOWN
User keeps on touching the panel	SEVT_MOUSE_LB_STILL_DOWN
User removes his hand/stylus from the panel	SEVT_MOUSE_LB_UP

Table 3 Input events from HAL

These events, along with the co-ordinates, when sent to the Sparklet, it will be processed and converted to proper actions such as button click, double click etc.

3.4.3 OS/Scheduler

The Sparklet library could be running on top of a bare metal system, or on an RTOS or even a full-fledged OS like Linux. And since Sparklet supports multiple applications at the same time, it is necessary for various mechanisms for sharing resources among execution contexts. HAL events are used to indicate occurrence of certain actions and HAL Locks are used for sharing resources exclusively. Following are the list of functions to be supported by the HAL in this aspect.

Function Name	Description
arch_init_event	Creates an event and returns the handle
arch_wait_for_event	Waits for the event on timeout in ms
arch_set_event	Sets the event
arch_clear_event	Clears the event
arch_delete_event	Deletes the event
arch_init_lock	Creates an lock event and returns the handle
arch_wait_for_lock	Waits and acquire the lock on timeout in ms
arch_release_lock	Releases the lock
arch_delete_lock	Closes the windows lock handle

Table 4 HAL Functions for internal usage

These mechanisms can be implemented using the native IPC methods for example, WaitForSingleObject in Win32, proprietary function in RTOS etc.

3.4.4 Timing

Timing is an important feature needed to determine the outcome of various possible actions. Apart of the functions, arch_wait_for_event and arch_wait_for_lock, which needs precise timing, Sparklet also calls sgui_time_delay_hmsm function internally. This must also be supported by the HAL.

3.5 Graphics Device Interface

Above the HAL, there is the functionality called Graphics Device Interface- GDI for short. It is the core layer that implements the functionality of the Sparklet. Now we will look in to some of the sub-modules that constitute the GDI.

3.5.1 Controls and Rendering

Every unit that occupies a space in the screen and that can handle an event is called a Control. Sparklet organizes these Controls in a linked list mechanism with Z ordering. A control may lie on top of one or more controls and might be visible fully or partially or not at all. Based on the visibility and ordering, Sparklet calculates the regions visible to the user and organizes them as draw regions. It is sufficient if only these draw regions are rendered as only these are visible. This calculation results in significance performance increase of the UI rendering.

3.5.2 Core Module

The core module is essentially the root window that is responsible for the overall functionality of the Sparklet library. The UI can be considered as a single window application where the top-most window also called the root window running on full screen. Any windows created by the user are technically child control of the root window.

This core module must be run as a separate execution task or as a thread, depending up on the underlying scheduling mechanism. It must call the ***init_sgui*** function followed by the ***handle_root_win_msg*** function before any other UI specific functions are called. This function will block on itself till the UI is stopped and handles the overall control of the library.

3.5.3 Event Handling

Event handling in Sparklet is done by a separate execution task or a thread. The function ***sgui_event_manager*** must be called from the thread. This function locks on itself till the library is exited and is responsible for handling various events provided into Sparklet via the functions like ***sevt_post_mouse_event***.

Once events are submitted, the Event handling module fetches them one by one from the queue and processes them. First, if the screen is rotated, the event co-ordinate is translated to the new one (landscape to portrait or portrait to landscape). Then the event is upgraded if needed. For example, a single click event will be upgraded to double click if another click has occurred in the immediate past. Then based on the co-ordinate the target control is found and the event handler of the same is called with the event.

Then the control that received the event can handle it as per the implementation requirements of the control.

3.5.4 Frame Buffer Handling

Sparklet uses very basic operations like drawing a pixel, a line, rectangle and filled rectangle to perform the entire necessary UI rendering. As the device screen might be rotated, the library does the necessary adjustment, and calls the architecture specific function to do these draw operations.

3.5.5 Image Rendering

Sparklet library, by itself supports rendering different kinds of images. Supported formats include BMP, RAW, PNG etc. Also these images could be loaded from memory or from a file. Refer to the Image APIs section of the Sparklet API manual for detailed description of associated functions.

3.5.6 Font Rendering

Fonts need to be compiled along with the Sparklet source. Fixed width fonts as well as variable width fonts are supported. Also it is possible to rotate the rendering to 90, 180 or 270 degrees counter clockwise. The functions supported can be found under the Text Operation APIs section of the Sparklet API Manual.

3.5.7 Messages

While events are mechanisms to provide the Sparklet inputs form HAL, messages are the mechanism used by Sparklet to send processed information to the User applications. Messages might be generated internally in response to a user interaction (Button action from Mouse Button Down) or timeout or user generated messages for inter application communication.

The messages are submitted asynchronously to the message queue of particular application and must be handled by the handler of the window.

3.6 Widgets

While the GDI layer provides all the necessary modules to work with the UI, Sparklet provides another layer consisting of modules called Widgets that are created for convenience of the user. Widgets are implemented on top of the control layer and provide a logical functionality in the UI like a button or a label. Widgets take care of creating underlying controls, rendering, event handling etc and finally output messages to the user layer. Some of the widgets available as of now are

- Button
- Canvas
- Check box
- Combo box
- Dialog
- Fixed view
- Graph
- Holder
- Image holder
- Key board

- List box
- List view
- Menu
- Meter
- Message box
- Progress bar
- Scrollbar
- Scroll view
- Slider
- Static
- Tab
- Taskbar
- Text Area
- Viewport
- Widget
- Window
- Text edit

For more details on how to use these widgets, please refer to the Sparklet API Manual. It is possible to create new widgets as per the user equipments as well. Implementation of widgets like Static, button and list box can be referred to implement custom widgets.

3.7 Applications

Applications represent organization of different functionalities as presented to the user. It is up to the user design philosophy to decide the way of organizing the applications. For example, in a HMI device, one application could be for configuration and another for viewing the current status.

In Sparklet, each application is created as a window that occupies the full screen. It supports running multiple applications at a time and it is possible to switch from one to another. But only one application can occupy the display at a time.

Chapter 4

Using Sparklet

This chapter introduces concepts for the developer and sets stage for developing application on Sparklet. Topics covered include code organization, configuration, building and running the library on target.

4.1 Code Organization

Source code of the Sparklet Embedded UI library is organized in a logical fashion to enable ease of navigation and modification. Written in 'C', there are two folders primarily – SRC and INC. the SRC folder contains the c files while the INC folder contains the header files. The following table provides the list of directories under these two directories.

Folder	Description
APP	User implementation of applications and other logic
FNT	Font Implementation
GDI	Graphics Device Interface implementation – the core modules of Sparklet
HAL	Hardware Abstraction Layer
UTIL	Utilities like memory pool allocator, PNG decoder etc
WID	Widgets and their implementation

Table 5 Directories and Organization

It is strongly recommended to follow this directory structure as it eases integration with other supporting tools like Windows Simulator and Flint IDE.

4.2 Configuration

Sparklet is highly configurable to suit the needs of the user. Different kinds of configuration options are available. Some of the major groups under which the options are organized are HAL Configuration, Library Configuration and Widget configuration. The details are explained in the following section.

4.2.1 HAL Configuration

Under this grouping, the options that are available can be used to customize the Sparklet for the underlying hardware platform. Configurations include setting the basic type of data types, architecture specific rendering operations

Some of the major configurations to be done for the Sparklet library for a particular device are

Option	Description
INT8U	Native data type for unsigned 8 bit data type
SCOLOR	Data type for color
SGUI_LOCK	Lock variable
SCOORD	Data type for co-ordinate

Table 6 HAL Configuration Options

More of these options are described in detail in the Sparklet API Manual.

4.2.2 Library Configuration

The necessary options to change the features available in the Sparklet library are grouped in this section. These options are sub-grouped as follows

- **General Configuration** – Setting display resolution, color depth, supported widgets etc.
- **Image Configuration** – Configuring list of supported image types etc.
- **Debug Configuration** – Debug options to be used during development.
- **Miscellaneous Configuration** –Other configurations like memory pool etc.

Some of the major configurations to be done for the Sparklet library for a particular device are

Option	Description
SGUI_DEFAULT_DISP_MODE	Default display mode
SGUI_DISPLAY_SCREEN_WIDTH	Display screen width
SGUI_DISPLAY_SCREEN_HEIGHT	Display screen height
VIRTUAL_SCREEN_WIDTH	Virtual screen width
VIRTUAL_SCREEN_HEIGHT	Virtual screen height
SGUI_MAX_NUM_APP_WINDOWS	Number of application windows
SGUI_DISABLE_TASKBAR	Enable/Disable taskbar
SGUI_SYSTEM_FONT	Default System Font

Table 7 Configuration Options

Detailed description of these configurations is documented in the Sparklet API Manual.

4.2.3 Widget Configuration

Various options are provided to configure the supported widgets including setting the size for internal components, supported actions etc. Refer to the Sparklet API Manual for complete list of the options.

4.3 Implementation

Users can start writing applications on top of Sparklet library with minimal effort. Various examples are provided in the documentation along with demo applications to start with. Tools like Flint are available to develop the UI screen graphically.

4.3.1 Starting Sparklet

Following is the typical sequence of steps to be followed when running the Sparklet.

1. Perform architecture specific initialization and set up RTOS.
2. Create a task to run the main functionality of Sparklet. Then call the ***init_sgui*** function followed by function ***handle_root_win_msg***
3. Create a task to run the event handling functionality of Sparklet. Then call the ***sgui_event_manager*** function form that thread.
4. Create custom tasks after these calls are executed. Sparklet calls sgui_start_user_app function, which can be used for this purpose.

The demo code provided will give a clear idea about starting Sparklet.

4.3.2 Threads and Priority

As explained, in a RTOS environment, each of the tasks can be run as separate threads. The following table provides an idea about the priority organization for the same.

Priority	
High	User specific non-UI task

Table 8 Task Priorities

4.4 Building

Building the library source is highly specific to the architecture and the development environment of the target. For more information for compiling the code, refer to the documentation of the same.

4.5 Running

Each target needs different procedure to run the code. Refer to the documentation of the associated tools/microcontroller for running the compiled output.

4.6 Simulation

The Sparklet library is provided with a Visual Studio 2008 project file to enable users develop their UI and test it on the host machine before deploying it on the target. Developing in a VS2008 environment significantly reduces the development time and helps catch many issues. An executable called SGUI.exe is generated up on compiling the same that can even be sent to different teams for validation purpose.

Chapter 5

APIs

This chapter provides information about knowing the Application Programmer Interface of the Sparklet Embedded GUI library

5.1 Available APIs

Sparklet UI library provides a clearly defined set of functions called Application Programmer Interface for the users to program with. These APIs are defined with intention of using the library seamlessly for different use cases.

The source code of the Sparklet UI is thoroughly documented. The comments sections are written for Doxygen and a manual called Sparklet API Manual is generated from the same. The latest version of that manual can be downloaded from the download section of our website – http:www.embien.com/downloads. Kindly refer to that manual for knowing about the functions and configuration options of the Sparklet.

Chapter 6

Customization

Sparklet provides numerous options to customize as per the user requirements. This chapter provides information about various possibilities like changing the Skin etc

6.1 Skin

It is possible to change the theme of the whole Sparklet library to match that of the customer's UI philosophy with very few modifications.

All the widgets are rendered using a function registered with it. The default implementation draws with a classic model. This default rendering can be simply disabled by enabling the following macros

DISABLE_CLASSIC_SKIN

When this macro is set to a non-zero value, the default functions are not compiled in. Then the user can implement function like sbutton_draw, sstatic_draw etc.

Typically these functions are of the form,

int sbutton_draw (SBUTTON *ptr_button, DRAW_INFO *ptr_draw_info)

The first argument provides the pointer to the base control. The second argument provides the pointer to the structure that provides information about the draw operation to be performed and has the following fields.

Field	Description
src_pos.x	X co-ordinate of the top-left position to draw the control
src_pos.y	Y co-ordinate of the top-left position to draw the control
ub_direct	Whether invoked directly or not (via a parent draw)
ub_draw_mode	Drawing mode
dst_pos	Destination position where it will be copied to. Must not be used.

Table 9 Draw Customization Information

Care should be taken to optimize the rendering by drawing only the draw-able-regions as much as possible. Also regions outside the given widget area must not be drawn.