

PocketBook SDK Documentation

Revision 1.05

1 Оглавление

2	Introduction	4
2.1	Minimal app(lication)	4
2.2	Getting real.....	5
3	How it works	7
3.1	InkViewMain()	7
3.2	main_handler()	8
3.3	Events delivered to main_handler().....	8
4	Output to a screen	10
4.1	ClearScreen()	10
4.2	SetClip()	10
4.3	DrawPixel()	10
4.4	DrawLine()	11
4.5	DrawRect().....	11
4.6	FillArea()	11
4.7	InvertArea()	11
4.8	InvertAreaBW().....	11
4.9	DimArea()	11
4.10	DrawSelection()	12
4.11	DitherArea()	12
4.12	Stretch()	12
4.13	SetCanvas()	13
4.14	GetCanvas().....	13
4.15	Repaint()	13
5	Screen update	13
5.1	FullUpdate().....	13
5.2	FullUpdateHQ().....	14
5.3	SoftUpdate()	14
5.4	PartialUpdate()	14
5.5	PartialUpdateBlack().....	14
5.6	PartialUpdateBW()	14
5.7	DynamicUpdate()	14
5.8	DynamicUpdateBW().....	14
5.9	FineUpdate().....	14
5.10	FineUpdateSupported()	14
5.11	HQUpdateSupported().....	14
5.12	ScheduleUpdate()	14

6	Bitmap handling.....	15
6.1	ibitmap structure.....	15
6.2	LoadBitmap()	15
6.3	SaveBitmap()	15
6.4	BitmapFromScreen()	15
6.5	BitmapFromScreenR()	15
6.6	NewBitmap()	15
6.7	LoadJPEG()	16
6.8	SaveJPEG()	16
7	Functions Reference	16
7.1	OpenScreen().....	16
8	Revision History	17

2 Introduction

2.1 Minimal app(lication)

```
// hello_world.cpp
#include "inkview.h"

static const int kFontSize = 40;

static int main_handler(int event_type, int /*param_one*/, int /*param_two*/) {
    if (EVT_INIT == event_type) {
        ifont *font = OpenFont("LiberationSans", kFontSize, 0);
        ClearScreen();
        SetFont(font, BLACK);
        DrawTextRect(0, ScreenHeight()/2 - kFontSize/2, ScreenWidth(),
                     kFontSize, "Hello, world!", ALIGN_CENTER);
        FullUpdate();
        CloseFont(font);
    } else if (EVT_KEYPRESS == event_type) {
        CloseApp();
    }

    return 0;
}

int main (int argc, char* argv[]) {
    InkViewMain(main_handler);

    return 0;
}
```

Command line to build

```
arm-none-linux-gnueabi-g++ hello_world.cpp -o hello_world -linkview
```

This simple app clears the screen and prints "Hello, world!" in the center of the screen. Pressing any¹ key exits the app.

Things to note:

1. One has to provide handler function in order to receive events from keyboard (EVT_KEYPRESS) and on app state changes (EVT_INIT).
2. Resources (font) acquired has to be released, when not needed any more: CloseFont() complements OpenFont().
3. Output to screen is performed in two steps:
 - a. Drawing into buffer: ClearScreen() and DrawTextRect().
 - b. Copying buffer or its part to screen: FullUpdate().Usually result of several drawing calls is send to screen with single update function.
4. In more realistic app one will need to have a set of data (variables) that describe application state and accessible to different functions (handlers) or on different invocations of the same handler. We will provide recommendations in the next subsection.
5. String "Hello, world!" is shown in the center of the screen. For vertical alignment calculations are performed manually. Horizontal alignment is done automatically as a result of passing ALIGN_CENTER flag to DrawTextRect() function.

¹ Exceptions will be explained later in section on key events.

2.2 Getting real

In this section structure for more realistic and thus complex app is suggested. This is one of many ways to organize the app, so you are free to adopt, adapt or ignore this suggestion.

As application grows the following should be addressed:

1. Separating and encapsulating application state.
2. Handling more events, while keeping `main_handler()` simple and straight forward.
3. Ability to configure app on initialization.
4. Cleanup on application exit.
5. Ability to handle key presses depending on key pressed.
6. Providing means to detect and handle errors.

// app_common.h

```
#ifndef APP_COMMON_H
#define APP_COMMON_H

const int kErrorFail = -1;

#define RETURN_ERROR_IF_NULL(ptr,error_code) {\
if (NULL == ptr) return error_code; }

#endif // APP_COMMON_H
```

// app_state.h

```
#ifndef APP_STATE_H
#define APP_STATE_H

#include <cstdint>

struct ifont_s;
typedef struct ifont_s ifont;

namespace HelloWorldTwo {

class AppState;
AppState& GetAppState();

class AppState {
public:
    // Constructor only sets initial values. No functions are called that may fail
    AppState()
        : font_(NULL)
        , font_size_(10)
    {}

    ~AppState() { Deinit(); }

    int OnInit(int font_size);
    int OnKeyPress(int key_code);
    int OnShow();
    void OnExit();

private:
    void Deinit();

    ifont *font_;
    int font_size_;
}; // class AppState

} // namespace HelloWorldTwo

#endif // APP_STATE_H
```

```

// app_state.cpp
#include "app_state.h"
#include "app_common.h"
#include "inkview.h"

namespace HelloWorldTwo {

AppState& GetAppState() {
    static AppState app_state;
    return app_state;
}

int AppState::OnInit(int font_size) {
    font_size_ = font_size;
    font_ = OpenFont("LiberationSans", font_size_, 0);
    RETURN_ERROR_IF_NULL(font_, kErrorFail);
    return 0;
}

int AppState::OnKeyPress(int /*key_code*/) {
    CloseApp();
    return 0;
}

int AppState::OnShow() {
    ClearScreen();
    SetFont(font_, BLACK);
    DrawTextRect(0, ScreenHeight()/2 - font_size_/2, ScreenWidth(),
                font_size_, "Hello, world!", ALIGN_CENTER);
    FullUpdate();
    return 0;
}

void AppState::OnExit() {
    Deinit();
}

void AppState::Deinit() {
    if (font_) {
        CloseFont(font_);
        font_ = NULL;
    }
}

} // namespace HelloWorldTwo

```

```

// hello_world_two.cpp
#include "app_state.h"
#include "inkview.h"

using namespace HelloWorldTwo;

static const int kFontSize = 40;

static int main_handler(int event_type, int param_one, int /*param_two*/) {
    int rv = 0;
    AppState& app_state = GetAppState();

    switch (event_type) {
    case EVT_INIT:
        rv = app_state.OnInit(kFontSize);
        break;

```

```

    case EVT_SHOW:
        rv = app_state.OnShow();
        break;

    case EVT_KEYPRESS:
        rv = app_state.OnKeyPress(param_one);
        break;

    case EVT_EXIT:
        app_state.OnExit();
        break;

    default:
        break;
}

return rv;
} // int main_handler(3)

int main (int argc, char* argv[]) {
    InkViewMain(main_handler);

    return 0;
}

```

Command line to build

```
arm-none-linux-gnueabi-g++ app_state.cpp hello_world_two.cpp -o hello_world_two -linkview
```

Things to note:

1. Header files don't include other header files unless absolutely necessary. `cstdint` is included because of `NULL`. This inclusion may be avoided by moving constructor to `.cpp` file.
2. Namespace is used for app specific classes and functions.
3. App state encapsulated into class (not struct!).
4. Handler calls methods of `AppState` class (rather than accessing struct members directly).
5. `AppState` instance is accessed via function, rather than using global variable. Time for `AppState` instance creation is definite: on the first `GetAppState()` call.
6. To pass `argc` and `argv` from `main()` to `AppState`, use method (not shown above) `AppState::SetArgs(int argc, char* argv[])` before calling `InkViewMain()`.

3 How it works

3.1 InkViewMain()

Simplest app contains at least two functions: `main()` from which `InkViewMain()` is called, and handler function, that is passed as a parameter to `InkViewMain()`.

Function's declaration:

```
typedef int (*iv_handler)(int event_type, int param_one, int param_two);

void InkViewMain(iv_handler main_handler);
```

`main_handler` – must be non-NULL, otherwise behavior is undefined.

`InkViewMain()` first performs initialization, then runs event loop. On exit from event loop cleanup is done, then function exits.

`OpenScreen()` is called on initialization.

3.2 main_handler()

Handler function passed to InkViewMain() has the following signature:

Syntax:

```
int main_handler(int event_type, int param_one, int param_two);
```

Parameters:

- event_type – event type, see EVT_* defines. Non-zero.
- param_one – meaning depends on event type;
- param_two – meaning depends on event type;

Returns:

Zero means that event wasn't handled by the app. In this case InkViewMain() will handle event. Non-zero means that event was handled by the app, and InkViewMain() does nothing for the event.

Notes:

Events that are handled by InkViewMain() on any firmware.

Event type	Description
EVT_INIT	Causes wake time to be at least SLEEPDELAY from now regardless of main_handler() return value.
EVT_PANEL_TASKLIST	Opens tasks list.
EVT_PANEL_MPLAYER	Opens media player, if not open, otherwise closes media player.
EVT_PANEL_BLUETOOTH	Opens Bluetooth info.
EVT_PANEL_BLUETOOTH_A2DP	Opens A2dp info.
EVT_PANEL_CLOCK	Opens calendar.

Events that are handled by InkViewMain() on firmware without multitasking support.

Event type	Description
EVT_KEYRELEASE	Short key press made, while not in menu, dialog or last open list: If param_one == KEY_MUSIC, opens media player. If param_one == KEY_POWER, performs action assigned to power key.
EVT_KEYPRESS	Opens main menu by home or menu key press, if "open main menu" action is assigned to key.
EVT_KEYREPEAT, KEY_MUSIC, 1	Toggles playing music.
EVT_KEYPRESS or EVT_KEYREPEAT	Increases (param_one == KEY_PLUS) or decreases (param_one == KEY_MINUS) playing volume.

3.3 Events delivered to main_handler()

The table below lists events delivered to main_handler() and parameters for events. In the table below the first column lists three values corresponding to parameters event_type, param_one and param_two. Second column explains under what circumstances event is called and what parameters mean.

Event type and parameters	Description
EVT_INIT, 0, 0	This is the first event delivered to main_handler(). Delivered only once per InkViewMain() call. Initialize application data on this event.

EVT_SHOW, 0, 0 EVT_REPAINT, 0, 0	<p>Delivered, when the app is required to draw itself:</p> <ul style="list-style-type: none"> • In InkViewMain() after EVT_INIT and before entering event loop. • For new handler, when setting new handler. <p>Note, that event isn't delivered after dialog or task manager window closed, or after switching back to the app. In such cases library saves app's screen content and then restores it.</p>
EVT_MP_STATECHANGED, new_state, 0	Delivered (to active task only) on detecting media player state change. new_state is one of MP_* values e.g. MP_PLAYING.
EVT_MP_TRACKCHANGED, new_track, 0	Delivered (to active task only) on detecting track change. new_track – currently playing track's number.
EVT_ORIENTATION, orientation, 0	<p>Delivered (to active task only) on detecting orientation change.</p> <p>Orientation – new orientation (0, 1, 2 or 3).</p> <p>0 – portrait</p> <p>1 – landscape = portrait turned 90° counterclockwise</p> <p>2 – landscape = portrait turned 90° clockwise</p> <p>3 – portrait = portrait turned 180°</p>
EVT_HIDE, 0, 0	<p>Delivered, when</p> <ul style="list-style-type: none"> • Exiting event loop in InkViewMain() for reason other than signal received. For example, calling CloseApp() from the app results in EVT_HIDE delivery, but signal SIGINT sent to the app - doesn't. • Setting new event handler. Event is delivered to the old handler.
EVT_EXIT, 0, 0	<p>Delivered as a result of:</p> <ul style="list-style-type: none"> • CloseApp() call from the app; • Receiving one of signals: SIGTERM, SIGINT, SIGQUIT or SIGHUP. <p>Isn't delivered:</p> <ul style="list-style-type: none"> • On performing "kill task" action, assigned to a key.
EVT_FOREGROUND, pid, 0	<p>Delivered (only if multitasking is present) to app, when it becomes active:</p> <ul style="list-style-type: none"> • On app start. • On switching to app (task). <p>pid – process ID of the app that receives the event</p>
EVT_BACKGROUND, pid, 0	<p>Delivered (only if multitasking is present) to app, when it becomes inactive i.e. on switching from app to another app (task).</p> <p>Note, that event isn't delivered on app exit. That's why number of EVT_BACKGROUND received is one less, than number of EVT_FOREGROUND.</p> <p>pid - process ID of the app that receives the event</p>
EVT_SNAPSHOT, 0, 0	Delivered to active app (task) in case app exits because of device shutdown e.g. result of power-off action. Delivered only if Configuration->Appearance->Boot logo is set to "current page". The app may create a screen snapshot and set it as boot logo, see ???.
EVT_KEYDOWN, key_code, 0	Delivered, when hardware key is pressed. key_code – code of key pressed, value of KEY_* macro, see inkview.h
EVT_KEYPRESS, key_code, 0	
EVT_KEYREPEAT, key_code, repeat_count	<p>Delivered periodically (now once in 0.5 sec) after hardware key is pressed and held longer than defined time period (now 1 sec).</p> <p>key_code – key code, value of KEY_* macro</p>

	repeat_count – number of times, starting with 1, EVT_KEYREPEAT is received for given key since EVT_KEYDOWN.
EVT_KEYUP, key_code, repeat_count	Delivered, when hardware key is released. key_code – code of released key, value of KEY_* macro
EVT_KEYRELEASE, key_code, repeat_count	repeat_count – number of times EVT_KEYREPEAT was called since last EVT_KEYDOWN. May be zero.
EVT_POINTERDOWN, x, y	Delivered, when stylus touches the screen. (x, y) – coordinates of touch point
EVT_POINTERMOVE, x, y	Delivered between EVT_POINTERDOWN and EVT_POINTERUP. The app should be ready to receive up to 25 events per second (in current implementation) (x, y) – current stylus coordinates
EVT_POINTERUP, x, y	Delivered, when stylus moved from screen. (x, y) – last known stylus coordinates.

4 Output to a screen

Output to screen involves two steps:

1. Drawing to current canvas which is buffer in memory.
2. Sending part or all of current canvas to screen driver. We will call this step "update", because functions that perform it has "Update" in their names.

All coordinates and sizes are in pixels, unless the opposite is said explicitly.

By default current canvas has dimensions of screen (orientation is respected).

4.1 ClearScreen()

Fills current canvas with **white** color.

```
void ClearScreen()
```

4.2 SetClip()

Sets clipping rectangle for current canvas. Drawing inside the rectangle changes canvas, drawing outside won't change canvas. Initially clipping rectangle is set to include all canvas.

```
void SetClip(int x, int y, int width, int height)
```

(x,y) – upper left corner of clipping rectangle. Must be ≥ 0 and \leq canvas width and height respectively.

width – must be positive

height – must be positive

Note: if parameters' values are out of range, result is undefined.

4.3 DrawPixel()

Draws pixel of given color in current canvas.

```
void DrawPixel(int x, int y, int color)
```

(x,y) – coordinates of pixel to draw. Must be within clipping rectangle for pixel to appear.

color – 24-bit color value of form 0x00RRGGBB. If canvas has depth 8 bits per pixel (bpp), then color is converted to 8-bit grayscale value, otherwise each color component is drawn separately.

Note, there is no alpha component.

Note: If canvas has depth other than 8 or 24 bpp, result is undefined.

4.4 DrawLine()

Draws straight line with ends in given points. Line is drawn as a series of pixels of given color. Line ends are included. Line isn't antialiased.

```
void DrawLine(int x_one, int y_one, int x_two, int y_two, int color)
```

(x_one, y_one) and (x_two, y_two) – coordinates of line ends.

color – 24-bit color value. See DrawPixel() for details.

4.5 DrawRect()

Draws rectangle of given color in current canvas. Rectangle has 1-pixel wide border and isn't filled inside. Lower right corner is (x+width-1, y+height-1).

```
void DrawRect(int x, int y, int width, int height, int color)
```

(x,y) – upper left corner

width – must be positive number

height – must be positive number

color - 24-bit color value. See DrawPixel() for details.

4.6 FillArea()

Draws filled rectangle of given color in current canvas. Rectangle has (x,y) and (x+width-1, y+height-1) as upper left and lower right corners, lower right corner included.

```
void FillArea(int x, int y, int width, int height, int color)
```

(x,y) – upper left corner

width – must be positive number

height – must be positive number

color - 24-bit color value. See DrawPixel() for details.

4.7 InvertArea()

Changes color value of each pixel within given rectangle with binary complement value. For example, grayscale pixel with color 0x16 becomes 0xE9. For 24-bit color pixels each component is binary complemented.

```
void InvertArea(int x, int y, int width, int height)
```

(x,y) – upper left corner

width – must be positive number

height – must be positive number

Note: result is undefined for canvas with depth different from 8 or 24.

4.8 InvertAreaBW()

For 8-bpp canvas effect is the same as InvertArea(). For 24-bpp color canvas each color component is replaced with grayscale value (all components of pixel have the same value) and binary complemented.

```
void InvertAreaBW(int x, int y, int width, int height)
```

4.9 DimArea()

Sets half of pixels inside given rectangle to given color. Changed pixels form chessboard like grid: only pixels that correspond to black chessboard squares are changed. In other words: pixel that has coordinates (xp, yp) is set to given color, if xp+yp is even, otherwise pixel is left unchanged.

```
void DimArea(int x, int y, int width, int height, int color)
```

4.10 DrawSelection()

Draws rounded rectangle with 3 pixel line width. Rectangle isn't filled and is inside given rectangle. Drawing is performed with given color and into current canvas.

```
void DrawSelection(int x, int y, int width, int height, int color)
```

4.11 DitherArea()

Function is used to decrease number of colors (intensity levels) to a value of "levels" parameter. To reduce negative effects of color quantization dithering can be used: method = DITHER_PATTERN or method = DITHER_DIFFUSION. Function is applied to given rectangle in current canvas. Input can be 8 bpp grayscale or 24 bit color. Result has same depth as input, but always grayscale (for color image all components have the same value).

```
void DitherArea(int x, int y, int width, int height, int levels, int method)
```

levels – number between 2 and 16 inclusive

method – can be one of:

DITHER_THRESHOLD
DITHER_PATTERN
DITHER_DIFFUSION

Usages:

- Call with levels=2 and method=DITHER_THRESHOLD to have image that can be quickly updated e.g. for menus, lists and selections.
- Call with levels=16 and method=DITHER_DIFFUSION to prepare 24-bit color image for drawing on 16-color screen.

4.12 Stretch()

Takes image, transforms it and draws in current canvas. Transformation includes rotation, scaling, mirroring or any combination of them.

```
void Stretch(const unsigned char* src, int src_format,  
             int src_width, int src_height, int src_line_size,  
             int dest_x, int dest_y, int dest_width, int dest_height, int flags)
```

src – buffer, where source image is located. Can't be NULL.

src_format – source image format. One of:

IMAGE_BW – 1 bpp image. Each source byte describes 8 pixels.

IMAGE_GRAY2 – 2 bpp image. Each source byte describes 4 pixels.

IMAGE_GRAY4 – 4 bpp image. Each source byte describes 2 pixels.

IMAGE_GRAY8 – 8 bpp image. Each source byte describes exactly 1 pixel.

Any other value – means 24-bit color image, where 3 source bytes describe 1 pixel.

src_width – width in pixels of source image. Must be positive.

src_height – height in pixels of source image. Must be positive.

(dest_x, dest_y) – position of upper left corner in the destination (current canvas).

dest_width – width of destination rectangle. Image will be scaled on x coordinate to fit/fill.

dest_height – height of destination rectangle. Image will be scaled on y coordinate to fit/fill.

flags – bit field. Two least significant bits specify rotation:

ROTATE0 – no rotation

ROTATE90 – rotate 90° counter clockwise

ROTATE270 – rotate 270° counter clockwise

ROTATE180 – stays for itself ☺

Flags may be OR-ed with XMIRROR and/or YMIRROR constant to mirror image relative x and y axis respectively.

Usages:

- May be used for scale image to fit/fill the screen.
- It is perfectly OK to have source and destination size the same and flags=0 to simply draw image on screen.

Notes:

- Flags affects clipping rectangle of canvas the same way as it affects image: clipping rectangle is rotated and/or mirrored.

4.13 SetCanvas()

Function sets given canvas to be a current canvas.

```
void SetCanvas(icanvas *new_canvas)

typedef struct icanvas_s {
    int width; /* in pixels */
    int height; /* in pixels */
    int scanline; /* size in bytes of single line (of pixels) */
    int depth; /* bits per pixel (bpp), can only be 8 or 24 */
    /* (clipx1, clipy1) and (clipx2, clipy2) are respectively coordinates
       in pixels of upper left and bottom right corner of clipping rectangle.
       One must ensure that clipping rectangle is inside canvas, otherwise
       result is undefined. */
    int clipx1, clipx2;
    int clipy1, clipy2;
    unsigned char *addr; /* buffer to store pixel data. Buffer must be at least
                           height*scanline bytes in size */
} icanvas;
```

Parameters:

new_canvas – if non-NULL, specifies a canvas to set. If NULL, default canvas (that of frame buffer) is set.

Usage: One can manually create canvas, pass it to SetCanvas() and then draw into it using such functions as DrawPixel() and FillArea().

4.14 GetCanvas()

Function returns current canvas.

```
icanvas* GetCanvas()
```

4.15 Repaint()

Put EVT_SHOW into app's events queue. Eventually EVT_SHOW will be delivered to main_handler()

```
void Repaint()
```

Usage: Call Repaint() to make app (eventually) redraw itself on the screen.

5 Screen update

5.1 FullUpdate()

Content of the whole screen buffer is sent to display driver. Display depth is set to 2 bpp (usually) or 4 bpp if necessary. Function isn't synchronous i.e. it returns faster, than display is redrawn. Update is performed for active app (task) only, if display isn't locked and NO_DISPLAY flag in ivstate.uiflags isn't set.

```
void FullUpdate()
```

Usage: Tradeoff between quality and speed. Recommended for text and common UI elements. Not recommended if quality of picture (image) is required, in such case use FullUpdateHQ().

5.2 FullUpdateHQ()

Deprecated. Use SoftUpdate() instead.

The only difference from FullUpdate() is that display depth is set to maximum: 4 bpp currently.

```
void FullUpdateHQ()
```

5.3 SoftUpdate()

Alternative to FullUpdate(). In effect is (almost) PartialUpdate() for the whole screen.

```
void SoftUpdate()
```

5.4 PartialUpdate()

Content of the given rectangle in screen buffer is sent to display driver. Function is smart and tries to perform the most suitable update possible: black and white update is performed if all pixels in given rectangle are black and white. Otherwise grayscale update is performed. If whole screen is specified (0, 0, ScreenWidth(), ScreenHeight()), then grayscale update is performed.

```
void PartialUpdate(int x, int y, int width, int height)
```

5.5 PartialUpdateBlack()

Deprecated. Use PartialUpdate() instead.

```
void PartialUpdateBlack(int x, int y, int width, int height)
```

5.6 PartialUpdateBW()

Deprecated. Use PartialUpdate() instead.

```
void PartialUpdateBW(int x, int y, int width, int height)
```

5.7 DynamicUpdate()

Deprecated. Use PartialUpdate() instead.

```
void DynamicUpdate(int x, int y, int width, int height)
```

5.8 DynamicUpdateBW()

Deprecated. Use PartialUpdate() instead.

```
void DynamicUpdateBW(int x, int y, int width, int height)
```

5.9 FineUpdate()

Deprecated. Use SoftUpdate() instead.

```
void FineUpdate()
```

5.10 FineUpdateSupported()

Deprecated. Returns non-zero if FineUpdate() is supported.

```
int FineUpdateSupported()
```

5.11 HQUpdateSupported()

Deprecated. Returns non-zero if FullUpdateHQ() is supported.

```
int HQUpdateSupported()
```

5.12 ScheduleUpdate()

Deprecated.

```
void ScheduleUpdate(int x, int y, int width, int height, int bw)
```

6 Bitmap handling

6.1 ibitmap structure

```
typedef struct ibitmap_s {
    unsigned short width;
    unsigned short height;
    unsigned short depth;
    unsigned short scanline;
    unsigned char data[];
} ibitmap;
```

6.2 LoadBitmap()

Loads bitmap from given BMP file. Input file must be uncompressed file with 16 or 256 colors. Input file's size can't exceed 1'500'000 bytes.

```
ibitmap *LoadBitmap(const char *filename);
```

filename – non-NULL string specifying path to BMP file to open.

Returns: pointer to newly created bitmap on success, NULL on failure. Returned bitmap has the same depth as display, 4 bpp currently.

Color R=128d,G=128d,B=64d in input file is transparent i.e. when bitmap is drawn on screen pixel with transparent color isn't drawn thus preserving current display pixel.

If transparent color is present, then returned bitmap has most significant bit (0x8000) of depth field set.

Use `DeleteBitmap()` to free memory used by bitmap.

6.3 SaveBitmap()

Saves given bitmap into file with given name. Format of output file is BMP with 4 or 16 colors, depending on `bm->depth`, that can be 2 or 4. **If file exists, it will be overwritten.** Output file is grayscale and doesn't have transparent color even if bitmap has one.

```
int SaveBitmap(const char *filename, ibitmap *bm);
```

Returns: non-zero on success, zero on failure.

6.4 BitmapFromScreen()

Same as `BitmapFromScreenR(x, y, width, height, 0)`.

```
ibitmap *BitmapFromScreen(int x, int y, int width, int height);
```

6.5 BitmapFromScreenR()

Function creates bitmap from the rectangular part of the current canvas. Bitmap is rotated as specified. Returned bitmap has the same depth as display, 4 bpp currently.

```
ibitmap *BitmapFromScreenR(int x, int y, int width, int height, int rotate);
```

rotate: 0 – don't rotate

1
2
3

Use `DeleteBitmap()` to free memory used by bitmap.

6.6 NewBitmap()

Creates 2 bpp bitmap with given dimensions.

```
ibitmap *NewBitmap(int width, int height);
```

6.7 LoadJPEG()

Loads image from given JPEG file into 2 bpp bitmap with given dimensions. Image is scaled to fit/fill given rectangle with or without preserving image's aspect ratio. Color correction is performed on intermediate 256 color grayscale image for each pixel using formula $\text{new_color} = (\text{current_color} - 128) * \text{co} / 100 + 128 + \text{br} - 100$. After this, intermediate image is converted into resulting 2 bpp bitmap.

```
ibitmap *LoadJPEG(const char *filename, int width, int height, int br, int co,
                  int proportional);
```

Parameters:

width, height – dimensions of resulting bitmap.

br, co – coefficients in color correction formula (see above). br = co = 100 – don't perform correction.

proportional – if non-zero, then aspect ratio for image is preserved, otherwise image's width and height may be changed independently in order to fit/fill dimensions of resulting bitmap.

6.8 SaveJPEG()

Saves given bitmap as JPEG file.

```
int SaveJPEG(const char *filename, ibitmap *bm, int quality);
```

Parameters:

filename – path to file, where to save bitmap. If file exists, it will be overwritten.

bm – 2 bpp or 4 bpp bitmap to save in file.

int – value between 0 and 100 inclusive. Higher quality values results in larger files.

Returns: non-zero on success, zero on failure.

7 Functions Reference

7.1 OpenScreen()

Function setups internal data structures as described below:

1. If not in safe mode, reads from GLOBALCONFIGFILE values for antialiasing, theme, font, font.b, font.i, font.bi, language, kbdlyaout.1{2,3}, keyboard, usbmode and showclock. If in safe mode, hardcoded defaults are used for mentioned values.
2. Reads from GLOBALCONFIGFILE values for keylock and poweroff (timeouts).
3. Sets actions for single and long key presses for power button and home, back, plus, minus, zoom-in and zoom-out keys (if present). Actions for short and long presses are read from values of form gkey.%n.0 and gkey.%n.1 respectively of GLOBALCONFIGFILE, where %n is key code. If not present in CONFIGFILE, hardcoded defaults are used, that depend on device model.
4. Gets access to frame buffer.
5. Updates orientation.
6. Loads strings for given language (see 1 above).
7. Opens theme.
8. Loads keyboard(s) for given language(s).
9. Allocates memory for (16) timers.
10. Allocates memory for (64) events queue.
11. For device build, obtains address of "OpenPlayer" ("_OpenPlayer") function in USERMPLAYER shared library. If not available, default internal function is used.

12. For device build, obtains address of "GetBookInfo" ("_GetBookInfo") and "GetBookCover" ("_GetBookCover") functions in USERBOOKINFO shared library. If not available, default internal functions are used.
13. Sets signal handlers for SIGHUP, SIGQUIT, SIGINT and SIGTERM.

Returns:

Always returns 1.

Usage:

This function is called from InkViewMain() before the first event is delivered to main_handler(). Thus avoid calling OpenScreen() explicitly from your app.

8 Revision History

Revision	Date	Comments
1.0	20-Jan-2012	The first revision
1.01	23-Jan-2012	Example in 1.1 finished. Example in 1.2 done.
1.02	7-Feb-2012	Frequently used events delivered to main_handler() are described.
1.03		Section "Output to screen" is in progress 95% complete.
1.04	13-Feb-2012	Section 5 "Screen update" is finished.
1.05	21-Feb-2012	Section 6 "Bitmap handling" in progress