Raw Pixel Image Format Proposal

Revision DRAFT1

ABSTRACT

This memo specifies the Raw Pixel Image (RPI) file format, a format used to hold image information and uncompressed pixel data in multiple pixel encodings. RPI format is designed to allow for the low processing overhead handling of pixel data to and from pixel based displays and input sources while providing a format compatible with editing and display applications. RPI format supports multiple pixels encodings, both with and without alpha channel.

STATUS OF THIS MEMO

This memo is a product of CLI Systems LLC engineering firm.

This memo provides information for the Internet community. Comments are solicited for discussion or improvements, and should be addressed to the author(s). Distribution of this memo is unlimited.

COPYRIGHT NOTICE

Table of Contents

INTRODUCTION

Purpose

The purpose of this specification is to define a file format designed to store image information and pixel data that:

- Is unencumbered by proprietary and/or undocumented vendor formats
- Accepts multiple pixel encoding formats
- Allows for low processing of pixel data to/from image sinks/sources
- Contains all image information related to size and encoding
- Portable format aligned on even boundaries for use in 8,16,32,and 64 bit systems
- Unencumbered by patents

Intended audience

This specification is intended for use by implementors of software to store, transfer, and display images in one of the supported pixel encodings.

This memo assumes a basic background in programming at the level of bits and other primitive data representations.

Scope

The specification specifies a file format to store image information. It does not specify any particular interface to file systems or character encoding.

Background

Uncompressed image formats are common in consumer devices, with manufacturer specific .RAW formats commonly used as ‘digital negatives’ to preserve the original image data before any lossy compression or color correction. While .RAW image formats are common for image sensors, there are many different types all proprietary, requiring proprietary software, leading to a lack of a standard open image format to store uncompressed (raw) pixel data.

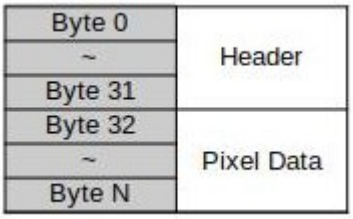
Common pixel formats include YUV, RGB565, RGB24, and various RGBA formats. Each of these formats contains uncompressed pixel information, YUV and RGB565 formats use 2 bytes (16bit) data, RGB24 is 3 bytes per pixel, and RGBA formats range from 16bit to 32bit.

Many modern cameras and LCD displays use uncompressed pixel based formats to transfer data in real-time with low processing overhead. USB and integrated cameras on laptop/desktop systems typically support the 32bit per 2byte (16bit per pixel) YUYV/UYUV formats. Embedded system cameras and LCDs can typically support RGB565, which allows for fast, low-processing overhead, direct camera-to-LCD and flash-to-LCD transfers.

FILE STRUCTURE:

The RPI image file format is designed to be lightweight and unobtrusive.

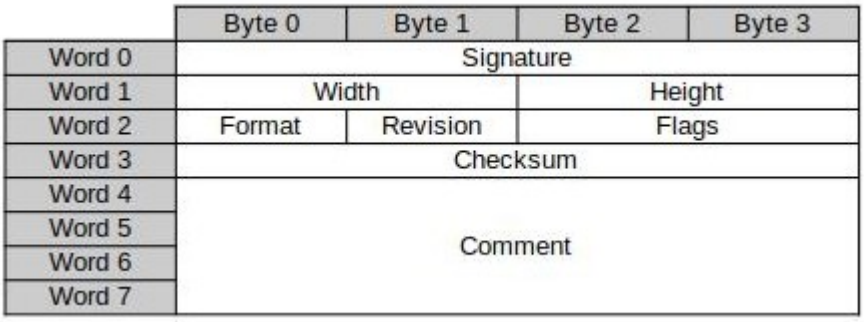
The structure consists of a 32 byte header followed by N bytes of pixel data. All information related to the pixel data is located in the header with no image information after pixel data




```
Byte 0   |------------|
         |            |
         | Header     |
         | 32-bytes   |
Byte 31  |            |
Byte 32  |------------|
         |            |
         | Pixel Data |
         | N-Bytes    |
         |            |
         |            |
Byte 31+N|------------|
```

Header structure:

The header structure consists of 8 fields for storing all information about the image including format, image dimensions, and user information.




```
       | Byte0  | Byte1  | Byte2  | Byte3  |
       ------------------------------------
Word 0 |            Signature              |
       ------------------------------------
Word 1 |      Width      |     Height      |
       ------------------------------------
Word 2 | Format | Rev.   |      Flags      |
       ------------------------------------
Word 3 |            Checksum               |
       ------------------------------------
Word 4 |            Comment                |
Word 5 |                                   |
Word 6 |                                   |
Word 7 |                                   |
       ------------------------------------
```

Header fields:

Name	Size	Index	Description
Signature	4 byte	0	Signature for file format, currently 0x52 0x50 0x49 0x31
Width	2 byte	4	Size of image width in pixels
Height	2 byte	6	Size of image height in pixels
Format	1 byte	8	Enumeration for pixel format
Revision	1 byte	9	File format revision number (default 0)
Flags	2 byte	10	Bitfield for flags for features
Checksum	4 byte	12	CRC checksum of payload
Comment	16 byte	16	User designed area, typically used for null terminated string data

Signature Field:

Used to provide a unique identifier for the file format

Signature: 0x52504931

Width & Height Field:

Uses to allow the fixed definition of the image data

Format Enumeration Field:

Multiple pixel based formats can be used with the RPI image format. The format enumeration field is used to indicate the bit packing structure of all pixel data.

Enumeration	Value	Description
FORMAT_RGB565	0x00	Red-Green-Blue format: Red 5bits, Green 6bits, Blue 5bits
FORMAT_BGR565	0x01	Blue-Green-Red format: Blue 5bits, Green 6bits, Red 5bits
FORMAT_YUYV	0x02	YUV 4-2-2 format: Y0 8bits, Cb 8bits, Y1 8bits, Cr 8bits
FORMAT_UYUV	0x03	UVY 4-2-2 format: Cb 8bits, Y0 8bits, Cr 8bits, Y1 8bits
FORMAT_RGAB5515	0x04	RGB+Alpha: Blue 5bits, Green 5bits, Alpha 1bit, Red 5bits
FORMAT_RGBA5551	0x05	RGB+Alpha: Blue 5bits, Green 5bits, Red 5bits, Alpha 1bit

See Appendix A – Pixel Encoding for details on individual encodings.

Revision Field:

Revision 0: Initial revision of the file format

Flags Field:

Flags can be used to control the processing of the pixel data.

Flag	Mask	Description
FLAG_CHECKSUM_ALL_DATA	0x0001	Checksum is for all data after the header Enabled – Checksum all data starting from Pixel Data (index 32) until the end of the file, including any data stored past the end of the pixel data Disabled – Checksum only data from Pixel Data start (index 32) until the end of the pixel data as calculated by the width, height, format values. Usages:

		- Prevents additional data from being concatenated to the file - Includes additional user data in the checksum for verification
FLAG_INVERT_PIXEL_DATA	0x0002	Indicate that pixel data represents the inverted value Enabled – Pixel data should be inverted for display Disabled – Pixel data represents the true value of the pixel Usages: Easy ‘reversal’ display, pixel data can be collected/stored ‘negative’ but displayed ‘positive’

Checksum Field:

The checksum is the CRC-32 calculation of all pixel data after the header. Pixels are all data from the end of the header until N; where N is width*height*[pixel enumeration width].

The polynomial of the CRC-32 checksum is 0x04C11DB7

CREDITS

Editor

Andrew Gaylo, admin@clisystems.com

Authors

Authors' names are presented in alphabetical order.

- Gaylo, Andrew - admin@clisystems.com

COPYRIGHT NOTICE

Authors' Contact

Andrew Gaylo
CLI Systems LLC
Lakewood, Colorado, USA

EMail: admin@clisystems.com

REFERENCES

[1] Introduction to graphics and LCD technologies (NXP) – https://www.nxp.com/wcm_documents/techzones/microcontrollers-techzone/Presentations/graphics.lcd.technologies.pdf – Slide 20

[2] Linux V4L2 Packed RGB Pixel Formats - https://www.kernel.org/doc/html/v4.8/media/uapi/v4l/pixfmt-packed-rgb.html

[3] Linux V4L2 Packed YUV Pixel Formats - https://www.kernel.org/doc/html/v4.8/media/uapi/v4l/pixfmt-packed-yuv.html

REVISION HISTORY

Revision	Date	Author	Notes
DRAFT1	August 1st 2021	CLI Systems LLC admin@clisystems.com	Initial revision

APPENDIX A – PIXEL ENCODINGS

FORMAT_RGB565 = 0x00

Pixels are packed in 5-6-5 format: Red 5bits, Green 6bits, Blue 5bits

BIT	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Data	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

Pseudo code:

```
uint16_t data;
uint8_t R = (data>>11)&0x1F
uint8_t G = (data>>5)&0x2F
uint8_t B = (data&0x1F)
```

FORMAT_BGR565 = 0x01

Pixels are packed in 5-6-5 format:
Structure: Blue 5bits, Green 6bits, Red 5bits

BIT	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Data	B4	B3	B2	B1	B0	G5	G4	G3	G2	G1	G0	R4	R3	R2	R1	R0

Pseudo code:

```
uint16_t data;
uint8_t B = (data>>11)&0x1F
uint8_t G = (data>>5)&0x2F
uint8_t R = (data&0x1F)
```

FORMAT_YUYV = 0x02

Pixels are packed in YUV 4-2-2 format, each four bytes is two pixels.
Structure: Y0 8bits, Blue Projection (Cb) 8bits, Y1 8bits, Red projection (Cr) 8bits
Pixel 0: Y0 Cb0 Cr0
Pixel 1: Y1 Cb0 Cr0

Byte	Byte0 (8bit)	Byte 1 (8bit)	Byte 2 (8bit)	Byte 3 (8bit)
Data	Luma 0	Blue Chrom.	Luma 1	Red Chrom.

BIT	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Data	Y07	Y06	Y05	Y04	Y03	Y02	Y01	Y00	Cb7	Cb6	Cb5	Cb4	Cb3	Cb2	Cb1	Cb0	Y17	Y16	Y15	Y14	Y13	Y12	Y11	Y10	Cr7	Cr6	Cr5	Cr4	Cr3	Cr2	Cr1	Cr0

FORMAT_UYUV = 0x03

Pixels are packed in UVY 4-2-2 format, each four bytes is two pixels.

Structure: Blue Projection (Cb) 8bits, Y0 8bits, Red projection (Cr) 8bits, Y1 8bits

Pixel 0: Y0 Cb0 Cr0

Pixel 1: Y1 Cb0 Cr0

Byte	Byte0 (8bit)	Byte 1 (8bit)	Byte 2 (8bit)	Byte 3 (8bit)
Data	Blue Chrom.	Luma 0	Red Chrom.	Luma 1

BIT	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Data	Cb7	Cb6	Cb5	Cb4	Cb3	Cb2	Cb1	Cb0	Y07	Y06	Y05	Y04	Y03	Y02	Y01	Y00	Cr7	Cr6	Cr5	Cr4	Cr3	Cr2	Cr1	Cr0	Y17	Y16	Y15	Y14	Y13	Y12	Y11	Y10

FORMAT_RGAB5515 = 0x04

Pixels are packed in 5-5-1-5 format:

Structure: Blue 5bits, Green 5bits, Alpha 1bit, Red 5bits

BIT	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Data	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	A	B4	B3	B2	B1	B0

Pseudo code:

```
uint16_t data;
uint8_t R = (data>>11)&0x1F
uint8_t G = (data>>6)&0x1F
uint8_t B = (data&0x1F)
bool A = ((data&0x20)==0x02);
```

FORMAT_RGBA5551 = 0x05

Pixels are packed in 5-5-5-1 format:

Structure: Blue 5bits, Green 5bits, Red 5bits, Alpha 1bit

BIT	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Data	R4	R3	R2	R1	R0	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0	A

Pseudo code:

```
uint16_t data;
uint8_t R = (data>>11)&0x1F
uint8_t G = (data>>6)&0x1F
uint8_t B = (data&0x3E)>>1
bool A = ((data&0x01)==0x01)
```

APPENDIX B – EXAMPLE SOURCE CODE

```
/**************************************
RPI File Format – Example Source Code
Copyright 2021 – CLI Systems LLC
admin@clisystems.com
**************************************/
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
    uint32_t signature;
    uint16_t width;
    uint16_t height;
    uint8_t format_enum;
    uint8_t revision;
    uint16_t flags;
    uint32_t checksum;
    int8_t comment[16];
    uint16_t payload[];
}IMG_header_t;

#define SIGNATURE_WORD          0x52504931

#define CHECKSUM_POLYNOMIAL     0x04C11DB7

typedef enum{
    FORMAT_RGB565   = 0x00,
    FORMAT_BGR565   = 0x01,
    FORMAT_YUYV     = 0x02,
    FORMAT_UYUV     = 0x03,
    FORMAT_RGBA5551 = 0x04,
    FORMAT_RGAB5515 = 0x05,
    FORMAT_RGB24    = 0x06,
}pixel_format_e;

#define FORMAT_RGB565_SIZE      sizeof(uint16_t)
```

```
#define FORMAT_BGR565_SIZE      sizeof(uint16_t)
#define FORMAT_YUYV_SIZE        sizeof(uint32_t)
#define FORMAT_UYUV_SIZE        sizeof(uint32_t)
#define FORMAT_RGBA5551_SIZE    sizeof(uint16_t)
#define FORMAT_RGAB5515_SIZE    sizeof(uint16_t)
#define FORMAT_RGB24_SIZE       (3)

#define FLAG_CHECKSUM_ALL_DATA     0x0001
#define FLAG_INVERT_PIXEL_DATA     0x0002

char * format_enum_to_string(pixel_format_e fmt)
{
    switch(fmt){
    case FORMAT_RGB565: return "FORMAT_RGB565";
    case FORMAT_BGR565: return "FORMAT_BGR565";
    case FORMAT_YUYV: return "FORMAT_YUYV";
    case FORMAT_UYUV: return "FORMAT_UYUV";
    case FORMAT_RGBA5551: return "FORMAT_RGBA5551";
    case FORMAT_RGAB5515: return "FORMAT_RGAB5515";
    case FORMAT_RGB24: return "FORMAT_RGB24";
    }
    return "FORMAT_UNKNOWN";
}
int format_enum_size(pixel_format_e fmt)
{
    switch(fmt){
    case FORMAT_RGB565: return FORMAT_RGB565_SIZE;
    case FORMAT_BGR565: return FORMAT_BGR565_SIZE;
    case FORMAT_YUYV: return FORMAT_YUYV_SIZE;
    case FORMAT_UYUV: return FORMAT_UYUV_SIZE;
    case FORMAT_RGBA5551: return FORMAT_RGBA5551_SIZE;
    case FORMAT_RGAB5515: return FORMAT_RGAB5515_SIZE;
    case FORMAT_RGB24: return FORMAT_RGB24_SIZE;
    }
    return 0;
}

uint32_t rpi_checksum(IMG_header_t * file)
{
```

```
    return 0;
}
void print_header(IMG_header_t * header)
{
    int payload_size;
    if(!header) return;
    printf("IMG_header_t\n");
    printf(" Signature  : 0x%08X\n",header->signature);
    printf(" Width      : %d\n",header->width);
    printf(" Height     : %d\n",header->height);
    printf(" Format Enum: 0x%02X (%s)\n",header->format_enum,format_enum_to_string(header->format_enum));
    printf(" revision   : %d\n",header->revision);
    printf(" Flags      : 0x%08X\n",header->flags);
    if(header->flags&FLAG_CHECKSUM_ALL_DATA) printf("  - FLAG_CHECKSUM_ALL_DATA\n");
    if(header->flags&FLAG_INVERT_PIXEL_DATA) printf("  - FLAG_INVERT_PIXEL_DATA\n");
    printf(" Checksum   : 0x%08X\n",header->checksum);
    printf(" Comment    : %s\n",header->comment);

    payload_size = header->width*header->height*format_enum_size(header->format_enum);

    printf(" Payload size: %d\n", payload_size);
    return;
}

int main(int cargc, char ** argv)
{
    uint16_t width,height;
    IMG_header_t * file;
    uint8_t *ptr;
    int size;
    int payload_size;
    pixel_format_e format;

    width=320;
    height=240;
    format = FORMAT_RGB565;

    payload_size = (width*height*format_enum_size(format));
```

```
    size = sizeof(IMG_header_t)+payload_size;
    ptr = (uint8_t*)malloc(size);
    file = (IMG_header_t*)ptr;

    // Setup header
    file->signature = SIGNATURE_WORD;
    file->width=width;
    file->height=height;
    file->format_enum = format;
    file->flags=0;
    file->checksum=0;
    snprintf(file->comment,sizeof(file->comment),"i16b Test1 4/21");

    // Fill image with data
    memset(file->payload,0,payload_size);

    // Calculate checksum
    file->checksum = rpi_checksum(file);

    printf("Running\n");
    printf("header size: %lu\n",sizeof(IMG_header_t));
    printf("buffer size: %d\n",size);

    print_header(file);

    free(ptr);

    return 0;
}
```