**PNG Signature**

First eight bytes are always:

137 80 78 71 13 10 26 10

**Chunk Layout**

Each chunk has

* **Length** – 4-byte unsigned int giving the number of bytes in the data field, does not include itself, the chunk type code, or the CRC. i.e. zero is a valid length.
* **Chunk Type** – 4-byte chunk type code. (code is restricted to A-Z and a-z)
* **Chunk Data** – the actual data bytes, can be of zero length.
* **CRC** – 4-byte Cyclic Redundancy Check. Always present even for empty chunks.

Chunks can appear in any order. (Apart from IHDR chunk must appear first, and IEND chunk must appear last.)

**Chunk Naming Conventions**

* Bit 5 of each byte convey chunk properties.
* **Ancillary Bit (bit 5, first byte)**
  + 0 = critical, 1 = ancillary
  + “ancillary” chunks can be ignored if the chunk code is unknown.
  + “critical” chunks are necessary successful displaying of the image.
  + If “critical” on an unknown chunk, we can not safely interpret the data.
* **Private Bit (bit 5, second byte)**
  + 0 = public, 1 = private
  + “public” chunks are part of the PNG specification, or in the list of special-purpose chunk types.
  + “private” chunks can be defined for custom purposes.
  + We do not need to test private-chunk property bits, as they have no functional significance.
* **Reserved Bit (bit 5, third byte)**
  + 0 = made using PNG specification 1.2
  + 1 = reserved for future specification, should be treated as an unknown chunk type.
* **Safe-To-Copy Bit (bit 5, fourth byte)**
  + 0 = unsafe to copy, 1 = safe to copy
  + Really only needed for apps that modify PNG files.

Example:

bLOb 🡨 32-bit chunk type code represented in text form.

|||+- safe-to-copy bit is 1.

||+-- reserved bit is 0.

|+--- private bit is 0.

+---- ancillary bit is 1.

Using this info, this name represents an ancillary, public, safe-to-copy chunk.

**Critical Chunks**

* **IHDR** - must be the first chunk of PNG file.
  + Width (4 bytes)
  + Height (4 bytes)
  + Bit depth (1 byte, values 1, 2, 4, 8, or 16)
  + Colour type (1 byte, values 0, 2, 3, 4, or 6)
  + Compression method (1 byte, value 0)
  + Filter method (1 byte, value 0)
  + Interlace method (1 byte, values 0 “no interface” or 1 “Adam7 interface”)
  + 13 bytes total
* **PLTE** – contains the palette, AKA list of colours.
  + Essential for colour type 3 (indexed colour)
  + Optional for colour types 2 & 6 (TrueColor & TrueColor with alpha)
  + Must **not** appear for colour types 0 & 4 (grayscale & grayscale with alpha)
* **IDAT** – contains the image.
  + Contains actual image data, which is the output of the compression algorithm. i.e. the chunks may not exactly match the PNG width\*height.
* **IEND** – marks end-of-file.
  + Has 0 data bytes / is empty.

**Ancillary Chunks**

* **bKGD** – default background colour.
* **cHRM** – chromaticity coordinates of the display primaries and white point.
* **cICP** – specifies colour space, transfer function and matrix coefficients. Intended for use with HDR’s without requiring colour profile.
* **dSIG** – stores digital signatures.
* **eXIf** – stores Exif metadata.
* **gAMA** – specifies gamma.
  + Contains only 4 bytes, representing gamma value multiplied by 100,000.
  + i.e. gamma value 1/3.4 => 29411.764… and is converted to an integer value of 29412 for storage.
* **hIST** – can store the histogram, or total amount of each colour in the image.
* **iCCP** – ICC colour profile.
* **iTXt** – contains a keyword and UTF-8 text.
* **pHYs** – holds intended pixel size / pixel aspect ratio. Contains**:**
  + Pixels-per-unit, X axis (4 bytes)
  + Pixels-per-unit, Y axis (4 bytes)
  + Unit specifier (1 byte)
* **sBit** – indicates colour-accuracy of the source data.
  + Contains 1-5 bytes, depending on the colour type.
* **sPLT** – suggested palette to use if the full range of colours is unavailable.
* **sRGB** – indicates standard sRGB colour space is used.
  + Contains 1 byte, used for “rendering intent”.
  + Values 0, 1, 2, 3 are defined for rendering intent.
* **sTER** – stereo-image indicator for stereoscopic images.
* **tEXt** – can store text, with one key-value pair for each chunk.
  + Key must be between 1-79 characters long; separator is null character.
  + Value can be any length, including zero.
* **tIME** – stores time the image was last changed / modified.
* **tRNS** – contains transparency information.
  + Indexed images = stores alpha channel values for one or more palette entries.
  + TrueColor / Grayscale images = stores a single pixel value that is to be regarded as fully transparent.
* **zTXt** – contains compressed text (and a compression method marker) with the same limits as **tEXt**.

The lower case first letter indicates that these chunks are not needed for PNG specs.

The lower case last letter indicates that the chunks are safe to copy, even if the application does not understand them.

**Pixel Format**

The ‘bits per channel’ in the table below is equal to the bit depth in the IHDR chunk.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Colour type** | **Channels** | **Bits per channel** | | | | |
| **1** | **2** | **4** | **8** | **16** |
| **Indexed** | **1** | 1 | 2 | 4 | 8 |  |
| **Grayscale** | **1** | 1 | 2 | 4 | 8 | 16 |
| **Grayscale + alpha** | **2** |  |  |  | 16 | 32 |
| **TrueColor** | **3** |  |  |  | 24 | 48 |
| **TrueColor + alpha** | **4** |  |  |  | 32 | 64 |

Pixels in PNG files are numbers that may be either indices of sample data in the palette, or the sample data itself. The palette is a separate table contained in the PLTE chunk.

Sample data for a pixel consists of a tuple of 1-4 numbers. These numbers are referred to as channels and every number in the image is encoded with an identical format.

Each number is encoded as an **unsigned int** using a fixed number of bits, referred to as the *bit depth*. This is not the same as the colour depth, which is the number of bits in each pixel.

The colour type is specified as an 8-bit value, however only the low three bits are used.

|  |  |
| --- | --- |
| 0 (0002­) | Grayscale |
| 2 (0102) | RGB / TrueColor |
| 3 (0112) | Indexed: channel containing indices into the palette of colours. |
| 4 (1002) | Grayscale + alpha: level of opacity for each pixel. |
| 6 (1102) | RGB / TrueColor + alpha |

**Chunk Layout**

|  |  |  |  |
| --- | --- | --- | --- |
| **Length** | **Chunk Type** | **Chunk Data** | **CRC** |
| 4 bytes | 4 bytes | *Length* bytes | 4 bytes |

**PLTE Chunk Layout**

* Contains 1-256 palette entries, each being in three-bytes:
  + Red: 1 byte (0 = black, 1 = red)
  + Green: 1 byte (0 = black, 1 = green)
  + Blue: 1 byte (0 = black, 1 = blue)
* Number of entries is chunk length / 3. i.e. 120/3 = 40 entries.
* The max number of entries is determined by 2^ (bit depth).
  + i.e. 2^8 = 256 max entries.
* The actual number of entries doesn’t have to reach the maximum, it just needs to be less than the max number of entries.

**IDAT Chunk Layout**

To read the raw chunk data:

1. De-compress the filtered data using the specified compression method.
2. Un-filter the image data using the specified filter method.
   1. The only filter method prepends a filter-type byte to each scanline.
3. The total size of the raw data is determined by the IHDR.
   1. The width and height in pixels.

There are three current pixel types supported:

1. *Indexed colour* – single sample that is an index into the supplied palette.
2. *Grayscale* – single sample that is the grayscale level
   1. Black = 0
   2. White = largest value for the bit depth
3. *TrueColor / RGB* – three samples (Red, Green, Blue).
   1. Bit depth specifies size of each sample, **not** total pixel size.

Optionally, *grayscale* and *RGB* can include an alpha channel as well.

Pixels smaller than a byte never cross byte boundaries, for example a 4-bit pixel will have 4 bits for colour/indexing and 4 bits for padding.

An extra “filter type” byte is added to the start of each scanline.

**Filter Byte Types**

0, None

* Filler byte, ignore.

1, Sub

* Difference between byte and corresponding byte of previous pixel.
  + Sub(x) = Raw(x) – Raw(x – bpp)
  + Assume first Raw(x - bpp) = 0

2, Up

* Difference between byte and corresponding byte of prior scanline (above byte)
  + Up(x) = Raw(x) – Prior(x)
  + Assume first pass Prior(x) = 0

3, Average

* Uses average of left and above pixels.
  + Average(x) = Raw(x) – floor( (Raw(x - bpp) + Prior(x)) / 2 )
  + Assume first Raw(x – bpp) = 0
  + Assume first pass Prior(x) = 0

4, Paeth

* Computes simple linear function of left, above, and upper left pixels.
  + Paeth(x) = Raw(x) – PaethPredictor(Raw(x – bpp), Prior(x), Prior(x – bpp))
  + PaethPredictor(a, b, c)

{

// a = left, b = above, c = upper left

p = a + b – c // initial estimate

pa = abs(p – a) // distances to a, b, c

pb = abs(p – b)

pc = abs(p – c)

if (pa <= pb && pa <= pc) return a

else if (pb <= pc) return b

else return c

}

* + Assume first Raw(x – bpp) = 0
  + Assume first pass Prior(x) = 0
  + Assume first Prior(x – bpp) = 0