

----------------------------------------------------------------------------------------------------------------------------------------

Obviously, the decoder operates with a sequence of bits received in a specific format. The binary stream is structured and divided into packets. On the upper level, there is separation of the stream on NAL-packets, and the stream has approximately the following form: 

|  |
| --- |
| [http://2.bp.blogspot.com/-vzjtICraVzE/TsPdnW_PV0I/AAAAAAAAAJs/5hEYkuEA_WI/s400/nal_stream.png](http://2.bp.blogspot.com/-vzjtICraVzE/TsPdnW_PV0I/AAAAAAAAAJs/5hEYkuEA_WI/s1600/nal_stream.png) |
| Figure 1. Stream separation on NAL-packets |

The abbreviation **NAL** stands for Network Abstraction Layer. The packet structure is shown in Figure 2.   
The first byte of a NAL-packet is a header that contains information about the type of packet. All the possible packet types are described in Table 1.   
  
Table 1. NAL types

|  |  |
| --- | --- |
| **Type** | **Definition** |
| 0 | Undefined |
| 1 | Slice layer without partitioning non IDR |
| 2 | Slice data partition A layer |
| 3 | Slice data partition B layer |
| 4 | Slice data partition C layer |
| 5 | Slice layer without partitioning IDR |
| 6 | Additional information (SEI) |
| 7 | Sequence parameter set |
| 8 | Picture parameter set |
| 9 | Access unit delimiter |
| 10 | End of sequence |
| 11 | End of stream |
| 12 | Filler data |
| 13..23 | Reserved |
| 24..31 | Undefined |

NAL-type defines what data structure is represented by current NAL-packet. It can be slice, or parameter set, or filler and so on. 

|  |
| --- |
| [http://4.bp.blogspot.com/-AA5KEs774LQ/TsPgkQVNPVI/AAAAAAAAAJ4/UPtNT9R2ZHA/s400/NAL_structure.png](http://4.bp.blogspot.com/-AA5KEs774LQ/TsPgkQVNPVI/AAAAAAAAAJ4/UPtNT9R2ZHA/s1600/NAL_structure.png) |
| Figure 2. NAL-packet structure |

As can be seen from the figure, the payload of NAL-packet identified as **RBSP** (Raw Byte Sequence Payload). RBSP describes a row of bits specified order of SODB (String Of Data Bits).  
So RBSP contains SODB. According to the ITU-T specification if SODB empty (zero bits in length), RBSP is also empty. The first byte of RBSP (most significant, far left) contains the eight bits SODB; next byte of RBSP shall contain the following eight SODB and so on, until there is less than eight bits SODB. This is followed by a stop-bits and equalizing bit (Figure 3) 

|  |
| --- |
| [http://1.bp.blogspot.com/-27v8dgAbqR0/TsPgyU8G_SI/AAAAAAAAAKE/8f-R26rWxKU/s400/RBSP_SODB.png](http://1.bp.blogspot.com/-27v8dgAbqR0/TsPgyU8G_SI/AAAAAAAAAKE/8f-R26rWxKU/s1600/RBSP_SODB.png) |
| Figure 3. Raw Byte Sequence Payload (RBSP) |

Now let’s look closer to our bitstream: 

|  |
| --- |
| [http://1.bp.blogspot.com/-XoBw_6CfYVc/TsPg6tlJoXI/AAAAAAAAAKQ/XfP2VJeJ6aE/s400/bitstream_detailed.png](http://1.bp.blogspot.com/-XoBw_6CfYVc/TsPg6tlJoXI/AAAAAAAAAKQ/XfP2VJeJ6aE/s1600/bitstream_detailed.png) |
| Figure 4. Detailed H.264 stream |

Any coded image contains slices, which in turn are divided into **macroblocks**. Most often, one encoded image corresponds to one slice. Also, one image can have multiple slices. The slices are divided into the following types:   
  
Table 2. Slice types

|  |  |
| --- | --- |
| **Type** | **Description** |
| 0 | P-slice. Consists of P-macroblocks (each macro block is predicted using one reference frame) and / or I-macroblocks. |
| 1 | B-slice. Consists of B-macroblocks (each macroblock is predicted using one or two reference frames) and / or I-macroblocks. |
| 2 | I-slice. Contains only I-macroblocks. Each macroblock is predicted from previously coded blocks of the same slice. |
| 3 | SP-slice. Consists of P and / or I-macroblocks and lets you switch between encoded streams. |
| 4 | SI-slice. It consists of a special type of SI-macroblocks and lets you switch between encoded streams. |
| 5 | P-slice. |
| 6 | B-slice. |
| 7 | I-slice. |
| 8 | SP-slice. |
| 9 | SI-slice. |

Looks like table 2 contains some redundant data, But that is not true: types 5 - 9 mean that all other slices of the current image will be the same type.   
  
As you noticed every slice consists of header and data. Slice header contains the information about the type of slice, the type of macroblocks in the slice, number of the slice frame. Also in the header contains information about the reference frame settings and quantification parameters. And finally the slice data – macroblocks. This is where our pixels are hiding.   
  
Macroblocks are the main carriers of information, because they contain sets of luminance and chrominance components corresponding to individual pixels. Without going into details it can be concluded that the video decoding is ultimately reduced to the search and retrieval of macroblocks out of a bit stream with subsequent restoration of pixels colors with help of luminance and chrominance components. This is how single macroblock looks like: 

|  |
| --- |
| [http://1.bp.blogspot.com/-kTrtXc7xxDQ/TsPh6nctEYI/AAAAAAAAAKc/dyScSoiZtXs/s400/macroblock.png](http://1.bp.blogspot.com/-kTrtXc7xxDQ/TsPh6nctEYI/AAAAAAAAAKc/dyScSoiZtXs/s1600/macroblock.png) |
| Figure 5. Macroblock |

Here we have macroblock type, **prediction type** (which is the subject of the next article), Coded Block Pattern, Quantization Parameter (if we have CPB) and finally – data: the sets of **luminance** and **chrominance** components.   
  
That is all for now. Next H.264 topic will be definitely dedicated to the macroblock prediction.   
I hope you enjoyed this article. Feel free to comment or ask any questions. Good luck.

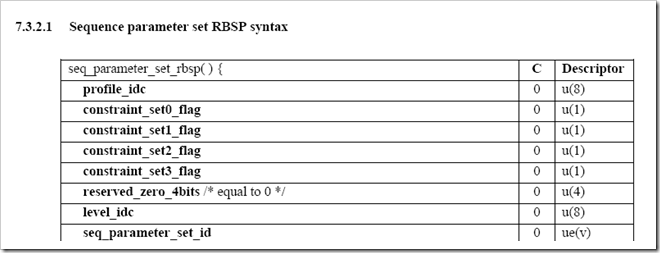
[**H.264 stream structure**](https://codesequoia.wordpress.com/2009/10/18/h-264-stream-structure/)

Posted on [October 18, 2009](https://codesequoia.wordpress.com/2009/10/18/h-264-stream-structure/)by [Moto](https://codesequoia.wordpress.com/author/motonariito/)

What you call “H.264 stream” is probably not the real raw H.264 stream – it’s probably “H.264 byte stream”. It’s important to use the proper terminology to discuss stream validity or multiplexing matters.

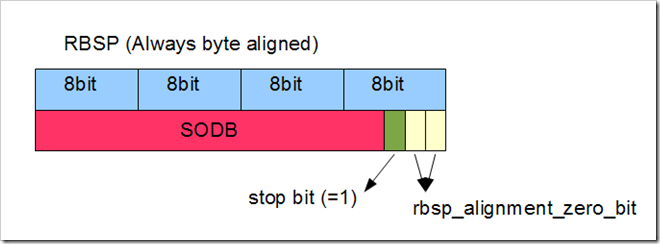
SODB

Let’s start from the SODB (String Of Data Bits). This is the real raw H.264 stream. The syntax is specified ISO/IEC 14496-10 in a form of bit string syntax.

[](http://nggewq.blu.livefilestore.com/y1pORF2xZSm6XF-oTCnkHFM6O2FH6-j1NeuWKkZRxy-n88LL3BqHcyGVwb5K0Bsjic5ld4nrlOmbAWlIMDzPXQQEcVMkhU4PFVD?PARTNER=WRITER)

RBSP

SODB is not very convenient because it’s a bit stream, not a byte stream. It means the number of bits for one syntax may not be byte-aligned and therefore difficult to process. This is why the standard introduces RBSP (Raw Byte Sequence Payload). RBSP stores SODB in a byte stream so that the first bit of each syntax is always aligned at the first bit of a byte.

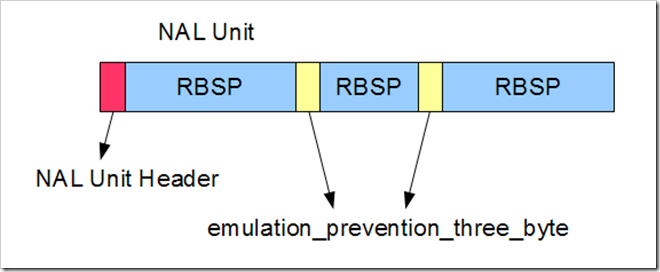
[](http://nggewq.blu.livefilestore.com/y1pt3SRYhnN7HUE2VSINzBb5W_zzlC0_WrGKB_pMOZqfUhNIMcc-hqP0f3Dcytf1TcS2NpOPEvAhDpSZCejRHeU7pNAfEiVDpBT?PARTNER=WRITER)

NAL Unit

The problem of RBSP is that RBSP may contain any byte pattern. Since it doesn’t have special synchronization byte sequence, to find the synchronization point in RBSP (For example, to find the first byte of IDR picture), you may have to parse every single bit syntax from the beginning of the file.

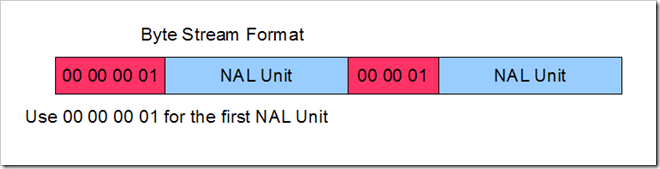
NAL Unit is another wrapper layer to prevent certain byte pattern from occurring in the stream. When RBSP has any of 0x000000, 0x000001, 0x000002, and 0x000003, they are converted to 0x00000300, 0x00000301, 0x00000302, and 0x00000303 respectively in NAL unit. Therefore, we can use any of 0x000000, 0x000001, 0x000002, or 0x000003 as a special synchronization byte sequence.

NAL Unit also adds one byte header to indicate the type of the NAL Unit.

[](http://blufiles.storage.msn.com/y1pftpRKqRWD4XurvFXDstrcLwcfxJG10gVuthRbnx7iRKnFTUoygefOcsZ3UhjjIUiUHJxVN-1_wY81i7rpT-Eyg?PARTNER=WRITER)

Byte Stream Format

Although NAL Unit allows to put a synchronization byte sequence, it doesn’t have any yet. The standard defines another wrapper to add three or four bytes synchronization byte pattern: the Byte Stream Format. The byte stream format puts a synchronization byte sequence (0x000001 or 0x00000001) before every NAL Unit. The byte stream format is used as the elementary stream of H.264 in transport stream.

[](http://nggewq.blu.livefilestore.com/y1pZPPgDl9mT1Pr0-uNlLLgzshvpbG6AObT3yxy5eGljWr33UTqxdqBXnKXCb7PlTZsUL0hDh_yCFdL6AyTJxeDnj7vFKnYeM3l?PARTNER=WRITER)

The original intention of the standard is to let applications to pick a suitable format. For example, MPEG Transport Stream uses byte stream format to allow decoder to find the  NAL Unit easily. Other container format such as AVI where the length of the header is stored in a packet would use NAL Unit or even RBSP to reduce the overhead of synchronization bytes. However, in real world, almost all storage and delivery formats use byte stream format. It’s due to practical reasons such as to simplify the re-wrapping process.

Nonetheless, to avoid any confusion, it’s better idea to use the term “H.264 Byte Stream Format” instead of “H.264 raw stream” or “H.264 stream”.