INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND ASSOCIATED AUDIO

ISO/IEC JTC1/SC29/WG11
MPEG96/M1138
Tampere, Finland

Title:	**MSDL-S Representation of the MPEG-2 Video IS**
Source:	**Yihan Fang and Alexandros Eleftheriadis (Columbia University)[1]**
Status:	**Proposal**
Group:	**MPEG-4 Integration/MSDL**
Date:	**July 4, 1996**

Introduction

This document is the complete MSDL-S representation of the bitstream syntax of the MPEG-2 Video International Standard [1]. The definition of MSDL-S used is as given in [2], with the addition of parameter types as detailed in [3]. For readability, we adopted the typographic convention of setting MSDL-S keywords in italics and bitstream elements in boldface.

The purpose of this document is to demonstrate, using a very familiar and non-trivial example, how MSDL-S can be used to express complex bitstream structures. Note that the Video_Sequence class has to be subclassed from the appropriate MSDL Architecture superclass. In order to obtain a complete MSDL-based specification of a functioning decoder, one only has to extend the class definitions to also include methods.

For simulation purposes, this MSDL-S file together with the associated methods (written in regular C++ or Java) can be translated to an equivalent C++ or Java one, in which bitstream parsing is handled within the object constructor(s). In addition to being important for testing and validating MSDL-S, this is also useful when writing decoder simulation software, as it automatically takes care of bitstream parsing. Essentially, the definition of the bitstream in MSDL-S can be directly used as part of a software implementation of its decoder. Such a translator is currently under implementation.

In an actual MPEG-4 environment, or its simulation, the bitstream specification is translated (by the same translator) to a binary format, and is downloaded to the decoder. Note that this downloading capability can be applied to just the syntax; in other words, the method declarations and definitions (tools) need not change. This allows flexibility in modifying the syntax even in Level 1 MPEG-4 systems.

[1] The authors can be contacted at: Dept. of Electrical Engineering, 1312 Mudd Bldg., Columbia University, 500 W. 120th Street, New York, NY 10027, USA, {fang,eleft}@itnm.columbia.edu. This document is also available at the URL: http://www.itnm.columbia.edu/~fang/MSDL/MPEG2.html

MPEG-2 Video

```
class Video_sequence {
   Sequence_header sh;
   if ([Sequence_extension.extension_start_code]){
      Sequence_extension se;
      do {
         Extension_and_user_data0 eud0;
         do {
            if ([Group_of_pictures_header.group_start_code]) {
               Group_of_pictures_header grp_of_ph;
               Extension_and_user_data1 eud1;
            }
            Picture_header ph;
            Picture_coding_extension pce;
            Extension_and_user_data2(se.progressive_sequence, pce.repeat_first_field,
                           pce.top_field_first, pce.picture_structure) eud2;
            Picture_data(se.chroma_format, sh.vertical_size_value,
                      se.vertical_size_extension, pce.f_code, pce.intra_vlc_format,
                      pce.picture_structure, pce.frame_pred_frame_dct,
                      eud0.ed, eud2.ed, ph.picture_coding_type) pd;
         } while ([Picture_header.picture_start_code]||
                [Group_of_pictures_header.group_start_code]);
         if (![sequence_end_code]) {
            Sequence_header sh;
            Sequence_extension se;
         }
      } while (![sequence_end_code]);
   } else {
      // ISO/IEC 11172-2
   }
   const bit(32) sequence_end_code = 0x000001B7;
};

class Sequence_header:
         const bit(32) sequence_header_code= 0x000001B3 {
   uint[2](12) horizontal_size_value;
   uint(12) vertical_size_value;
   uint(4) aspect_ratio_information;
   uint(4) frame_rate_code;
   uint(18) bit_rate_value;
   bit(1) marker_bit;
   uint(10) vbv_buffer_size_value;
   bit(1) constrained_parameters_flag;
   uint(1) load_intra_quantiser_matrix;
```

[2] For convenience, in this document, we use *uint* to denote *unsigned int* type.

```
    if ( load_intra_quantiser_matrix )
        uint(8*64) intra_quantiser_matrix[64];
    uint(1) load_non_intra_quantiser_matrix;
    if ( load_non_intra_quantiser_matrix )
        uint(8*64) non_intra_quantiser_matrix[64];
};


class Extension_and_user_data0 {
    while ([Sequence_extension.extension_start_code] ||
            [User_data.user_data_start_code]){
        if ([Sequence_extension.extension_start_code])
            Extension_data0 ed;
        if  ([User_data.user_data_start_code])
            User_data ud;
    };
};

class Extension_and_user_data1 {
    while ([Sequence_extension.extension_start_code] ||
            [User_data.user_data_start_code]){
        if  ([User_data.user_data_start_code])
            User_data ud;
    };
};

class Extension_and_user_data2(uint progressive_sequence,
                                int repeat_first_field,
                                uint top_field_first,
                                uint picture_structure) {
    while ([Sequence_extension.extension_start_code] ||
            [User_data.user_data_start_code]){
        if ([Sequence_extension.extension_start_code])
            Extension_data2(progressive_sequence, repeat_first_field,
                    top_field_first, picture_structure) ed;
        if  ([User_data.user_data_start_code])
            User_data ud;
    };
};

class Extension_data0 {
    while [Sequence_extension.extension_start_code] {
        bit(32) extension_start_code=Sequence_extension.extension_start_code;
        // follows Sequence_extension
        if  ([Sequence_display_extension.extension_start_code_identifier])
            Sequence_display_extension sde;
        else
```

```
        Sequence_scalable_extension sse;
    }
};

class Extension_data2(uint progressive_sequence,
                      uint repeat_first_field,
                      uint top_field_first,
                      uint picture_structure) {
    while [Sequence_extension.extension_start_code] {
        bit(32) extension_start_code=Sequence_extension.extension_start_code;
        // follows picture_coding_extension
        if  ([Quant_matrix_extension.extension_start_code_identifier])
            Quant_matrix_extension qme;
        else if ([Copyright_extension.extension_start_code_identifier])
            Copyright_extension ce;
        else if ([Picture_display_extension.extension_start_code_identifier])
            Picture_display_extension(progressive_sequence, repeat_first_field,
                                top_field_first, picture_structure) pde;
        else if ([Picture_spatial_scalable_extension.extension_start_code_identifier])
            Picture_spatial_scalable_extension psse;
        else
            Picture_temporal_scalable_extension ptse;
    }
};

class User_data:
        const bit(32) user_data_start_code = 0x000001B2{
    while (![0b0000.0000.0000.0000.0000.0001])
        uint(8) user_data;
};

class Sequence_extension:
        const bit(32) extension_start_code = 0x000001B5{
    const uint(4) extension_start_code_identifier=0b0001;
    uint(8) profile_and_level_indication;
    uint(1) progressive_sequence;
    uint(2) chroma_format;
    uint(2) horizontal_size_extension;
    uint(2) vertical_size_extension;
    uint(12) bit_rate_extension;
    bit(1) marker_bit;
    uint(8) vbv_buffer_size_extension;
    uint(1) low_delay;
    uint(2) frame_rate_extension_n;
    uint(5) frame_rate_extension_d;
};
```

```
class Sequence_display_extension {
    const uint(4) extension_start_code_identifier = 0b0010;
    uint(3) video_format;
    uint(1) colour_description;
    if (colour_description) {
        uint(8) colour_primaries;
        uint(8) transfer_characteristics;
        uint(8) matrix_coefficients;
    }
    uint(14) display_horizontal_size;
    bit(1) marker_bit;
    uint(14) display_vertical_size;
};

class Sequence_scalable_extension {
    const bit(32) extension_start_code_identifier = 0b0101;
    uint(2) scalable_mode;
    uint(4) layer_id;
    if (scalable_mode == 1 ) {                      // spatial scalability
        uint(14) lower_layer_prediction_horizontal_size;
        bit(1) marker_bit;
        uint(14) lower_layer_prediction_vertical_size;
        uint(5) horizontal_subsampling_factor_m;
        uint(5) horizontal_subsampling_factor_n;
        uint(5) vertical__subsampling_factor_m;
        uint(5) vertical__subsampling_factor_n;
    }
    if (scalable_mode == 3){                        // temporal scalability
        uint(1) picture_mux_enable;
        if (picture_mux_enable)
            uint(1) mux_to_progressive_sequence;
        uint(3) picture_mux_order;
        uint(3) picture_mux_factor;
};

class Group_of_pictures_header:
        const bit(32) group_start_code = 0x000001B8 {
    bit(25) time_code;
    uint(1) closed_gop;
    uint(1) broken_link;
};

class Picture_header:
        const bit(32) picture_start_code = 0x00000100 {
    uint(10) temporal_reference;
    uint(3)  picture_coding_type;
```

```
  uint(16) vbv_delay;
  if (picture_coding_type ==2 || picture_coding_type ==3) {
    bit(1) full_pel_forward_vector=0;
    bit(3) forward_f_code=0b111;
  }
  if (picture_coding_type ==3) {
    bit(1) full_pel_backward_vector=0;
    bit(3) backward_f_code=0b111;
  }
  while ([1]) {
    uint(1) extra_bit_picture;
    uint(8) extra_information_picture;
  }
  uint(1) extra_bit_picture;
};

class Picture_coding_extension {
  const bit(32) extension_start_code = Sequence_extension.extension_start_code;
  const bit(4) extension_start_code_identifier = 0b1000;
  uint(4*4) f_code[2][2];       // [0][0],[0][1],[1][0],[1][1]
                                // forward horizontal,forward vertical
                                // backward horizontal, backward vertical
  uint(2) intra_dc_precision;
  uint(2) picture_structure;
  uint(1) top_field_first;
  uint(1) frame_pred_frame_dct;
  uint(1) concealment_motion_vectors;
  uint(1) q_scale_type;
  uint(1) intra_vlc_format;
  uint(1) alternate_scan;
  uint(1) repeat_first_field;
  uint(1) chroma_420_type;
  uint(1) progressive_frame;
  uint(1) composite_display_flag;
  if ( composite_display_flag ) {
    uint(1) v_axis;
    uint(3) field_sequence;
    uint(1) sub_carrier;
    uint(7) burst_amplitude;
    uint(8) sub_carrier_phase;
  }
};


class Quant_matrix_extension {
  const bit(4) extension_start_code_identifier = 0b0011;
  uint(1) load_intra_quantser_matrix;
```

```
    if (load_intra_quantiser_matrix)
        uint(8*64) intra_quantiser_matrix[64];
    uint(1) load_non_intra_quantser_matrix;
    if (load_non_intra_quantiser_matrix)
        uint(8*64) non_intra_quantiser_matrix[64];
    uint(1) load_chroma_intra_quantser_matrix;
    if (load_chroma_intra_quantiser_matrix)
        uint(8*64) chroma_intra_quantiser_matrix[64];
    uint(1) load_chroma_non_intra_quantser_matrix;
    if (load_chroma_non_intra_quantiser_matrix)
        uint(8*64) chroma_non_intra_quantiser_matrix[64];
};

class Picture_display_extension (uint progressive_sequence,
                                 uint repeat_first_field,
                                 uint top_field_first,
                                 uint picture_structure) {
    const bit(4) extension_start_code_identifier = 0b0111;
    // now calculating $n: number_of_frame_centre_offsets
    if (progressive_sequence==1) {
        if (repeat_first_field==1) {
            if (top_field_first==1) $n=3;
            else $n=2;
        else $n=1;
    } else {
        if (picture_structure==0b01 ||        // field
            picture_structure==0b10)
            $n=1;
        else {
            if (repeat_first_field==1) $n=3;
            else $n=2;
        }
    }
    for ($i=0; $i<$n; $i++ ){
        int(16) frame_centre_horizontal_offset;
        bit(1) marker_bit;
        int(16) frame_centre_vertical_offset;
        bit(1) marker_bit;
    }
};

class Picture_temporal_scalable_extension {
    const bit(4) extension_start_code_identifier = 0b1010;
    uint(2) reference_select_code;
    uint(10) forward_temporal_reference;
    bit(1) marker_bit;
    uint(10) backward_temporal_reference;
```

```
};

class Picture_spatial_scalable_extension {
   const bit(4) extension_start_code_identifier = 0b1001;
   uint(10) lower_layer_temporal_reference;
   bit(1) marker_bit;
   int(15) lower_layer_horizontal_offset;
   bit(1) marker_bit;
   int(15) lower_layer_vertical_offset;
   uint(2) spatial_temporal_weight_code_table_index;
   uint(1) lower_layer_progressive_frame;
   uint(1) lower_layer_deinterlaced_field_select;
};

class Copyright_extension {
   bit(4) extension_start_code_identifier = 0b0100;
   bit(1) copyright_flag;
   uint(8) copyright_identifier;
   bit(1) original_or_copy;
   uint(7) reserved;
   bit(1) marker_bit;
   uint(20) copyright_number_1;
   bit(1) marker_bit;
   uint(22) copyright_number_2;
   bit(1) marker_bit;
   uint(22) copyright_number_3;
};

class Picture_data(uint chroma_format,
                   uint vertical_size_value,
                   uint vertical_size_extension,
                   int f_code[2][2],
                   uint intra_vlc_format,
                   uint picture_structure,
                   uint frame_pred_frame_dct,
                   Extension_data0 ed0,
                   Extension_data2 ed2,
                   uint picture_coding_type) {
   do {
      Slice(vertical_size_value, vertical_size_extension, ed0.sse,
           ed2.psse, picture_coding_type, f_code, intra_vlc_format,
           picture_structure, frame_pred_frame_dct) slice;
   }while([Slice.slice_start_code]);
};

class Slice(uint vertical_size_value,
            uint vertical_size_extension,
```

```
                Sequence_scalable_extension sse,
                Picture_spatial_scalable_extension psse,
                uint picture_coding_type,
                int f_code[2][2],
                uint intra_vlc_format,
                uint picture_structure,
                uint frame_pred_frame_dct)
    const bit (32) slice_start_code = 0x00000101..0x000001AF {
  if ((vertical_size_value +
      vertical_size_extension<< 12) > 2800)
    uint(3) slice_vertical_position_extension;
  if (sse.parsed) {
    if (sse.scalable_mode == 0)        // data partitioning
      uint(7) priority_breakpoint;
  }
  uint(5) quantiser_scale_code;
  if ([1]) {
    bit(1) intra_slice_flag;
    uint(1) intra_slice;
    uint(7) reserved_bits;
    while ([1]){
      uint(1) extra_bit_slice;    // with the value '1'
      uint(8) extra_information_slice;
    }
  }
  uint(1) extra_bit_slice;           // with the value '0'
  do {
    Macroblock(chroma_format, sse, psse, picture_coding_type,
            f_code, intra_vlc_format, picture_structure,
            frame_pred_frame_dct) macroblock;
  } while (![0b0000.0000.0000.0000.0000.0000.0001])
};

class Macroblock(uint chroma_format, Sequence_scalable_extension sse,
                  Picture_spatial_scalable_extension psse,
                  uint picture_coding_type,
                  int f_code[2][2],
                  uint intra_vlc_format,
                  uint picture_structure,
                  uint frame_pred_frame_dct) {
  while ([0b0000.0001.000]);
    bit(11) macroblock_escape;
  map(vlc_table_B1) macroblock_address_increment;
  Macroblock_modes(sse, psse, picture_coding_type, picture_structure,
              frame_pred_frame_dct) mb_modes;
  if ( mb_modes.macroblock_type == 1)     // macroblock_quant
    uint(5) quantiser_scale_code;
```

```
    if ( mb_modes.macroblock_type.macroblock_motion_forward ||
        ( mb_modes.macroblock_type.macroblock_intra &&
          Picture_coding_extension.concealment_motion_vectors) )
        Motion_vectors(0, mb_modes.motion_vector_count, mb_modes.mv_format,
                    mb_modes.dmv, f_code) motion_vectors_0;
    if ( mb_modes.macroblock_type.macroblock_motion_backward )
        Motion_vectors(1, mb_modes.motion_vector_count, mb_modes.mv_format,
                    mb_modes.dmv, f_code) motion_vectors_1;
    if ( mb_modes.macroblock_type.macroblock_intra &&
         Picture_coding_extension.concealment_motion_vectors )
        bit(1) marker_bit;
    if ( mb_modes.macroblock_type.macroblock_pattern )
        Coded_block_pattern(chroma_format,
                    mb_modes.macroblock_type.macroblock_intra) cbp;

    if (chroma_format == 1) {           // 4:2:0, block_count=6
        for ($i=0; $i<6; $i++)
            Block($i, mb_modes.macroblock_type.macroblock_intra,
                intra_vlc_format, cbp.pattern_code) block;
    } else if (chroma_format == 2 ) {   // 4:2:2, block_count=8
        if ($i=0; $i<8; $i++)
            Block($i, mb_modes.macroblock_type.macroblock_intra,
                intra_vlc_format, cbp.pattern_code) block;
    } else if (chroma_format == 3 ) {   // 4:4:4, block_count=12
        for ($i=0; $i<12; $i++)
            Block($i, mb_modes.macroblock_type.macroblock_intra,
                intra_vlc_format, cbp.pattern_code) block;
    }
};

class Macroblock_modes(Sequence_scalable_extension sse,
                       Picture_spatial_scalable_extension psse,
                       uint picture_coding_type,
                       uint picture_structure,
                       uint frame_pred_frame_dct) {
    // Reference table 6-10 in H.262
    // these cases use macroblock_types table B-2, B-3, B-4
    if ((!sse.parsed) ||                // not present in the bitstream
        sse.scalable_mode==0 ||         // data partitioning
        (sse.scalable_mode==1 && (!psse.parsed)) || // spatial scalability
        sse.scalable_mode==3)           // temporal scalability
        if (picture_coding_type==0b001)     // I picture
            map(vlc_table_B2) macroblock_type;
        if (picture_coding_type==0b010)     // P picture
            map(vlc_table_B3) macroblock_type;
        if (picture_coding_type==0b011)     // B picture
            map(vlc_table_B4) macroblock_type;
```

```
    // these cases use macroblock_types table B-5, B-6, B-7
    } else if (sse.scalable_mode==0b01 && psse.parsed) { // spatial scalability
        if (picture_coding_type==0b001)     // I picture
            map(vlc_table_B5) macroblock_type;
        if (picture_coding_type==0b010)     // P picture
            map(vlc_table_B6) macroblock_type;
        if (picture_coding_type==0b011)     // B picture
            map(vlc_table_B7) macroblock_type;
    // this case use macroblock_types table B-8
    } else if (sse.scalable_mode==2)        // SNR scalability
        map(vlc_table_B8) macroblock_type;

    if ( (macroblock_type.spatial_temporal_weight_code_flag == 1) &&
           (psse.spatial_temporal_weight_code_table_index != 0)) {
        int(2) spatial_temporal_weight_code;
    }
    if ( macroblock_type.macroblock_motion_forward ||
         macroblock_type.macroblock_motion_backward ) {
        if ( picture_structure == 0b11 ) {  // Frame
            if ( frame_pred_frame_dct == 0 )
                map(table6_17) frame_motion_type;
                int motion_vector_count=frame_motion_type.motion_vector_count;
                int mv_format=frame_motion_type.mv_format;
                int dmv=frame_motion_type.dmv;
        } else {
            map(table6_18) field_motion_type;
            int motion_vector_count=field_motion_type.motion_vector_count;
            int mv_format=frame_motion_type.mv_format;
            int dmv=field_motion_type.dmv;
        }
    }
    if ( ( picture_structure == 0b11 ) &&  // Frame Picture
         ( frame_pred_frame_dct == 0 ) &&
         ( macroblock_type.macroblock_intra ||
           macroblock_type.macroblock_pattern) ){
        uint(1) dct_type;
    }
};

// frame_motion_type
map table6_17 (uint(2), int motion_vector_count, int mv_format, int dmv) {
    0b00     -1     int     -1,
    0b01      2      0       0,   // mv_format 0:field 1:frame
    0b01      1      0       0,
    0b10      1      1       0,
    0b11      1      0       1
};
```

```
// field_motion_type
map table6_18 (uint(2), int motion_vector_count, int mv_format, int dmv) {
    0b00     -1      int     -1,
    0b01      1       0       0,
    0b10      2       0       0,
    0b11      1       0       1
};


class Motion_vectors(int s, int motion_vector_count, int mv_format, int dmv,
                            uint f_code[2][2]) {
    // motion_vector_count is determined by frame_motion_type or field_motion_type
    if (motion_vector_count==1) {
        if (( mv_format == 0 ) && (dmv !=1))     // mv_foramt==field
                uint(1) motion_vertical_field_select[0][s];
        Motion_vector(0,s,f_code,dmv) motion_vector_0;
    } else {
        uint(1) motion_vertical_feield_select[0][s];
        Motion_vector(0,s,f_code,dmv) motion_vector_0;
        uint(1) motion_vertical_feield_select[1][s];
        Motion_vector(1,s,f_code,dmv) motion_vector_1;
    }
};

class Motion_vector(int r, int s, uint f_code[2][2], int dmv) {
    map(vlc_table_B10) motion_code[r][s][0];  // 1-11 bits
    if (( f_code[s][0] !=1 ) && ( motion_code[r][s][0] !=0 ))
        uint(f_code[s][0]-1) motion_residual[r][s][0];
    if (dmv == 1)
        map(vlc_table_B11) dmvector[0];      // 1-2 bits
    map(vlc_table_B10) motion_code[r][s][1];
    if (( f_code[s][1] != 1) && (motion_code[r][s][1] !=0 ))
        uint(f_code[s][1]-1) motion_residual[r][s][1]; // r_size=f_code[s][t]-1
    if ( dmv == 1 )
        map(vlc_table_B11) dmvector[1];      // 1-2 bits
};

class Coded_block_pattern(uint chroma_format, uint macroblock_intra) {
    map(vlc_table_B9) coded_block_pattern_420; // 3-9 bits
    if ( chroma_format == 0b10) // 4:2:2
        uint(2) coded_block_pattern_422;
    if ( chroma_format == 0b11) // 4:4:4
        uint(6) coded_block_pattern_444;

    // calculate pattern_code[i]
    uint pattern_code[12];
```

```
    for ($i=0; $i<12; $i++) {
        if (macroblock_intra)
            pattern_code[$i]=1;
        else
            pattern_code[$i]=0;
    }
    if (macroblock_pattern) {
        for ($i=0; $i<6; $i++)
            if (coded_block_pattern_420 &(1<<(5-$i))) pattern_code[$i]=1;
        if (chroma_format == 0b10)     // 422
            for ($i=6; $i<8; $i++)
                if (coded_block_pattern_1 &(1<<(7-$i))) pattern_code[$i]=1;
        if (chroma_format == 0b11)     // 444
            for ($i=8; $i<12; $i++)
                if (coded_block_pattern_2 &(1<<(11-$i))) pattern_code[$i]=1;
    }
};

class Block(int i, uint macroblock_intra, uint intra_vlc_format, uint pattern_code[12]) {
    if ( pattern_code[i] ) {
        if ( macroblock_intra ) {
            if ( i<4 ) {
                map(vlc_table_B12) dct_dc_size_luminance;  // 2-9 bits
                if ( dct_dc_size_luminance !=0 )
                    uint(dct_dc_size_luminance) dct_dc_differential;
            } else {
                map(vlc_table_B13) dct_dc_size_chrominance;
                if ( dct_dc_size_chrominance !=0 )
                    uint(dct_dc_size_chrominance) dct_dc_differential;
            }
        } else {
            // First DCT coefficient 2-24 bits
                map(vlc_table_B14_first) first_DCT_coeff;
        }
        // Subsequent DCT coefficient
        if (macroblock_intra && intra_vlc_format){  // Table B-14
            while (![0b10*])      // End of Block
                map(vlc_table_B15) sub_DCT_coeff;
        }else{
            while(![0b0110*])     // End of Block
                map(vlc_table_B14_subsequent) sub_DCT_coeff;
        }
    }
};


// macroblock addressing variable length code table
```

```
// mapping macroblock_address_increment VLC code to increment value
map vlc_table_B1 (vlc, int increment_value) {
    0b1                     1,
    0b011                   2,
    0b010                   3,
    0b0011                  4,
    0b0010                  5,
    0b0001.1                6,
    0b0001.0                7,
    0b0000.111              8,
    0b0000.110              9,
    0b0000.1011            10,
    0b0000.1010            11,
    0b0000.1001            12,
    0b0000.1000            13,
    0b0000.0111            14,
    0b0000.0110            15,
    0b0000.0101.11         16,
    0b0000.0101.10         17,
    0b0000.0101.01         18,
    0b0000.0101.00         19,
    0b0000.0100.11         20,
    0b0000.0100.10         21,
    0b0000.0100.011        22,
    0b0000.0100.010        23,
    0b0000.0100.001        24,
    0b0000.0100.000        25,
    0b0000.0011.111        26,
    0b0000.0011.110        27,
    0b0000.0011.101        28,
    0b0000.0011.100        29,
    0b0000.0011.011        30,
    0b0000.0011.010        31,
    0b0000.0011.001        32,
    0b0000.0011.000        33,
    0b0000.0001.000        -1          //MACROBLOCK_ESCAPE
};

// Table B-2: Variable length codes For macroblock_type in I-pictures
map vlc_table_B2 (vlc, uint macroblock_quant,
                  uint macroblock_motion_forward,
                  uint macroblock_motion_backward,
                  uint macroblock_pattern, uint macroblock_intra,
                  uint spatial_temporal_weight_code_flag,
                  int permitted_spatial_temporal_weight_classes) {

    0b1          0   0   0   0   1   0   0,    // intra
```

```
    0b01           1   0   0   0   1   0   0     // intra, quant
};

// Table B-3: Variable length codes For macroblock_type in P-pictures
map vlc_table_B3 (vlc, uint macroblock_quant,
                         uint macroblock_motion_forward,
                         uint macroblock_motion_backward,
                         uint macroblock_pattern, uint macroblock_intra,
                         uint spatial_temporal_weight_code_flag,
                         int permitted_spatial_temporal_weight_classes) {

    0b1            0   1   0   1   0   0   0,    // MC, coded
    0b01           0   0   0   1   0   0   0,    // no MC, coded
    0b001          0   1   0   0   0   0   0,    // MC, not coded
    0b0001.1       0   0   0   0   1   0   0,    // Intra
    0b0001.0       1   1   0   1   0   0   0,    // MC, Coded, quant
    0b0000.1       1   0   0   1   0   0   0,    // no MC, coded, quant
    0b0000.01      1   0   0   0   1   0   0     // intra, quant
};

// Table B-4 Variable length codes For macroblock_type in B-pictures
map vlc_table_B4 (vlc, uint macroblock_quant,
                         uint macroblock_motion_forward,
                         uint macroblock_motion_backward,
                         uint macroblock_pattern, uint macroblock_intra,
                         uint spatial_temporal_weight_code_flag,
                         int permitted_spatial_temporal_weight_classes) {

    0b10           0   1   1   0   0   0   0,    // Interp, Not Coded
    0b11           0   1   1   1   0   0   0,    // Interp, Coded
    0b010          0   0   1   0   0   0   0,    // Bwd, Not Coded
    0b011          0   0   1   1   0   0   0,    // Bwd, Coded
    0b0010         0   1   0   0   0   0   0,    // Fwd, Not Coded
    0b0011         0   1   0   1   0   0   0,    // Fwd, Coded
    0b0001.1       1   0   0   0   1   0   0,    // Intra
    0b0001.0       1   1   1   1   0   0   0,    // Interp, Coded, Quant
    0b0000.11      1   1   0   1   0   0   0,    // Fwd, Coded, Quant
    0b0000.10      1   0   1   1   0   0   0,    // Bwd, Coded, Quant
    0b0000.01      1   0   0   0   1   0   0     // Intra, Quant
};

// Table B-5 VLC For macroblock_type in I-pictures with spatial scalability
map vlc_table_B5 (vlc, uint macroblock_quant,
                         uint macroblock_motion_forward,
                         uint macroblock_motion_backward,
                         uint macroblock_pattern, uint macroblock_intra,
                         uint spatial_temporal_weight_code_flag,
```

```
                        int permitted_spatial_temporal_weight_classes) {

   0b1          0  0  0  1  0  0  4,   // Coded, Compatible
   0b01         1  0  0  1  0  0  4,   // Coded, Compatible, Quant
   0b0011       0  0  0  0  1  0  0,   // Intra
   0b0010       1  0  0  0  1  0  0,   // Intra, Quant
   0b0001       0  0  0  0  0  0  4,   // Not Coded, Compatible
};

// Table B-6 VLC For macroblock_type in P-pictures with spatial scalability
map vlc_table_B6 (vlc, uint macroblock_quant,
                        uint macroblock_motion_forward,
                        uint macroblock_motion_backward,
                        uint macroblock_pattern, uint macroblock_intra,
                        uint spatial_temporal_weight_code_flag,
                        int permitted_spatial_temporal_weight_classes) {

   0b10         0  1  0  1  0  0   0,   // MC, Coded
   0b011        0  1  0  1  0  1  -1,   // MC, Coded, Compatible
   0b0000.100   0  0  0  1  0  0   0,   // No MC, Coded
   0b0001.11    0  0  0  1  0  1  -1,   // No MC, Coded, Compatible
   0b0010       0  1  0  0  0  0   0,   // MC, Not Coded
   0b0000.111   0  0  0  0  1  0   0,   // Intra
   0b0011       0  1  0  0  0  1  -1,   // MC, Not Coded, Compatible
   0b010        1  1  0  1  0  0   0,   // MC, Coded, Quant
   0b0001.00    1  0  0  1  0  0   0,   // No MC, Coded, Quant
   0b0000.110   1  0  0  0  1  0   0,   // Intra, Quant
   0b11         1  1  0  1  0  1  -1,   // MC, Coded, Compatible, Quant
   0b0001.01    1  0  0  1  0  1  -1,   // No MC, Coded, Compatible, Quant
   0b0001.10    0  0  0  0  0  1  -1,   // No MC, Not Coded, Compatible
   0b0000.101   0  0  0  1  0  0   4,   // Coded, Compatible
   0b0000.010   1  0  0  1  0  0   4,   // Coded, Compatible, Quant
   0b0000.011   0  0  0  0  0  0   4    // Not Coded, Compatible
};

// Table B-7 VLC For macroblock_type in B-pictures with spatial scalability
map vlc_table_B7 (vlc, uint macroblock_quant,
                        uint macroblock_motion_forward,
                        uint macroblock_motion_backward,
                        uint macroblock_pattern, uint macroblock_intra,
                        uint spatial_temporal_weight_code_flag,
                        int permitted_spatial_temporal_weight_classes) {

   0b10      0  1  1  0  0  0  0,   // Interp, Not Coded
   0b11      0  1  1  1  0  0  0,   // Interp, Coded
   0b010     0  0  1  0  0  0  0,   // Back, Not Coded
   0b011     0  0  1  1  0  0  0,   // Back, Coded
```

```
    0b0010        0   1   0   0   0   0   0,    // Fwd, Not Coded
    0b0011        0   1   0   1   0   0   0,    // Fwd, Coded
    0b0001.10     0   0   1   0   0   1   -1,   // Back, Not Coded, Compatible
    0b0001.11     0   0   1   1   0   1   -1,   // Back, Coded, Compatible
    0b0001.00     0   1   0   0   0   1   -1,   // Fwd, Not Coded, Compatible
    0b0001.01     0   1   0   1   0   1   -1,   // Fwd, Coded, Compatible
    0b0000.110    0   0   0   0   1   0   0,    // Intra
    0b0000.111    1   1   1   1   0   0   0,    // Interp, Coded, Quant
    0b0000.100    1   1   0   1   0   0   0,    // For, Coded, Quant
    0b0000.101    1   0   1   1   0   0   0,    // Back, Coded, Quant
    0b0000.0100   1   0   0   0   1   0   0,    // Intra, Quant
    0b0000.0101   1   1   0   1   0   1   -1,   // For, Coded,Compatible,Quant
    0b0000.0110.0 1   0   1   1   0   1   -1,   // Back, Coded,Compatible,Quant
    0b0000.0111.0 0   0   0   0   0   0   4,    // Not Coded, Compatible
    0b0000.0110.1 1   0   0   1   0   0   4,    // Coded,Compatible,Quant
    0b0000.0111.1 0   0   0   1   0   0   4     // Coded,Compatible
};

// Table B-8 VLC For macroblock_type in I-, P- and B-pictures with SNR scalability
map vlc_table_B8 (vlc, uint macroblock_quant,
                  uint macroblock_motion_forward,
                  uint macroblock_motion_backward,
                  uint macroblock_pattern, uint macroblock_intra,
                  uint spatial_temporal_weight_code_flag,
                   int permitted_spatial_temporal_weight_classes) {

    0b1           0   0   0   1   0   0   0,    // Coded
    0b01          1   0   0   1   0   0   0,    // Coded, Quant
    0b001         0   0   0   0   0   0   0,    // Not Coded
};

// Table B-9: Variable length codes For coded_block_pattern
map vlc_table_B9 (vlc, int cbp) {
    0b111            60,
    0b1101            4,
    0b1100            8,
    ...
    0b0000.0000.1       0       // note, not used with 420
};

// Table B-10: Variable length codes For motion_code
map vlc_table_B10 (vlc, int motion_code) {
    0b0000.0011.001     -16,
    0b0000.0011.011     -15,
    0b0000.0011.101     -14,
    ...
    0b0000.0011.000      16
```

```
};

// Table B-11: Variable length codes For dmvector[t]
map vlc_table_B11 (vlc, int value) {
   0b11         -1,
   0b0           0,
   0b10          1
};

// Table B-12: Variable length codes For dct_dc_size_luminance
map vlc_table_B12 (vlc, int dct_dc_size_luminance) {
   0b100              0,
   0b00               1,
   0b01               2,
   0b101              3,
   0b110              4,
   0b1110             5,
   0b1111.0           6,
   0b1111.10          7,
   0b1111.110         8,
   0b1111.1110        9,
   0b1111.1111.0      10,
   0b1111.1111.1      11
};

// Table B-13: Variable length codes For dct_dc_size_chrominance
map vlc_table_B13 (vlc, int dct_dc_size_chrominance) {
   0b00               0,
   0b01               1,
   0b10               2,
   0b110              3,
   0b1110             4,
   0b1111.0           5,
   0b1111.10          6,
   0b1111.110         7,
   0b1111.1110        8,
   0b1111.1111.0      9,
   0b1111.1111.10     10,
   0b1111.1111.11     11
};

// Table B-14 DCT coefficients Table zero: first DCT coefficient
map vlc_table_B14_first(vlc, int run, int level) {
   0b10               -1      int,   // End of block
   0b1s                0      1,     // used for the first coefficient
   0b011s              1      1,
   0b0100s             0      2,
```

```
   0b0101s              2     1,
   0b0010.1s            0     3,
   ...
   0b0000.0000.0001.1011s  31     1
};
```

```
// Table B-14 DCT coefficients Table zero: subsequent DCT coefficient
map vlc_table_B14_subsequent(vlc, int run, int level) {
   0b10                -1     int,    // End of block
   0b11s                0     1,      // used for all other coefficients
   0b011s               1     1,
   0b0100s              0     2,
   0b0101s              2     1,
   0b0010.1s            0     3,
   ...
   0b0000.01         uint(6)  int(12),  // ESCAPE
   ...
   0b0000.0000.0001.1011s  31     1
};
```

```
// Table B-15 DCT coefficients Table one
map vlc_table_B15(vlc, int run, int level) {
   0b0110              -1     int,    // End of block
   0b10s                0     1,
   0b010s               1     1,
   0b110s               0     2,
   0b0010.1s            2     1,
   0b0111s              0     3,
   ...
   0b0000.01         uint(6)  int(12),  // ESCAPE
   ...
   0b0000.0000.0001.1011s  31     1
};
```

References

[1] MPEG-2 Video. Recommendation ITU-T H.262 (1995 E), ISO/IEC 13818-2 International Standard, 1995.

[2] MSDL Specification. Version 1.1, ISO/IEC JTC1/SC29/WG11 N1246, Florence, Italy, March 1996.

[3] A. Eleftheriadis and Y. Fang, Adding Parameter Types to MSDL-S, ISO/IEC JTC1/SC29/WG11 M1055, Tampere, Finland, July 1996.