Technical Specification Group Services and System Aspects ***TSGS#19(03)0085***
Meeting #19, Birmingham, UK, 17 - 20 March 2003

Source: **TSG-SA WG4**

Title: CR to TS 26.073 - MMS compatible input/output option (Release 5)

Document for: **Approval**

Agenda Item: **7.4.3**

The following CR, agreed at the TSG-SA WG4 meeting #25bis, is presented to TSG SA #19 for approval.

Spec	CR	Rev	Phase	Subject	Cat	Vers	WG	Meeting	S4 doc
26.073	017		Rel-5	MMS compatible input/output option	F	5.0.0	S4	TSG-SA WG4#25bis	S4-030141

3GPP TSG-SA4 Meeting #25bis
Berlin, Germany, 24-28 February 2003

Tdoc ⌘S4-030141

CR-Form-v7

CHANGE REQUEST

⌘ **26.073** **CR** **017** ⌘**rev** **-** ⌘ Current version: **5.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ [ ] ME [X] Radio Access Network [ ] Core Network [X]

Title:	⌘	MMS compatible input/output option
Source:	⌘	TSG SA WG4
Work item code:	⌘	AMR
Date:	⌘	18/03/2003
Category:	⌘	**F**
Release:	⌘	Rel-5

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
Rel-4	*(Release 4)*
Rel-5	*(Release 5)*
Rel-6	*(Release 6)*

Reason for change:	⌘	Modifications proposed by this document enable usage of AMR fixed-point to encode and decode files according to the AMR MIME file storage format, which is used e.g. by the MMS service.
Summary of change:	⌘	New input/output option.
Consequences if not approved:	⌘	Codec can not operate with bitstreams in AMR MIME file storage format

Clauses affected:	⌘	2, 6.3, source files “coder.c”, “sp_enc.c”, “sp_enc.h”, “decoder.c”, “sp_dec.c”, “sp_dec.h”, “bitno.tab”

Other specs affected:	⌘	Y	N	
			X	Other core specifications
			X	Test specifications
			X	O&M Specifications

⌘

Other comments:	⌘	

Changes to the specification document:

2. References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.

[1] TS 26.074: "AMR Speech Codec; Test sequences".

[2] TS 26.090: "AMR Speech Codec; Speech transcoding".

[3] TS 26.091: "AMR Speech Codec; Substitution and muting of lost frames".

[4] TS 26.092: "AMR Speech Codec; Comfort noise aspects".

[5] TS 26.093: "AMR Speech Codec; Source controlled rate operation".

[6] TS 26.094: "AMR Speech Codec; Voice Activity Detection".

[7] RFC 3267 “A Real-Time Transport Protocol (RTP) Payload Format and File Storage Format for Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs, June 2002.

6.3 Parameter bitstream file (encoder output / decoder input)

The files produced by the speech encoder/expected by the speech decoder contain an arbitrary number of frames in the following format.

FRAME_TYPE	B1	B2	…	B244	MODE_INFO	*unused1*	…	*unused4*

Each box corresponds to one `Word16` value in the bitstream file, for a total of 250 words or 500 bytes per frame. The fields have the following meaning:

`FRAME_TYPE` transmit frame type, which is one of

`TX_SPEECH`	`(0x0000)`
`TX_SID_FIRST`	`(0x0001)`
`TX_SID_UPDATE`	`(0x0002)`
`TX_NO_DATA`	`(0x0003)`

`B0…B244` speech encoder parameter bits (i.e. the bitstream itself). Each `B`*x* either has the value `0x0000` or `0x0001`. Only mode `MR122` really uses all 244 bits; for the other modes, only the first *n* bits are used ($35 \le n \le 204$). The remaining bits are unused (written as `0x0000`)

`MODE_INFO` encoding mode information, which is one of

`MR475`	`(0x0000)`

```
MR515          (0x0001)
MR59           (0x0002)
MR67           (0x0003)
MR74           (0x0004)
MR795          (0x0005)
MR102          (0x0006)
MR122          (0x0007)
```

unused1…4 unused, written as `0x0000`

As indicated in section 6.1 above, the byte order depends on the host architecture.

By using a preprocessor definition the encoder output and decoder input can optionally use format described in [7], sections 5.1 and 5.3.

Changes to the c source-code:

Changes in file *coder.c*

Lines 53 – 56:

```
#ifdef MMS_IO
#define AMR_MAGIC_NUMBER "#!AMR\n"
#define MAX_PACKED_SIZE (MAX_SERIAL_SIZE / 8 + 2)
#endif
```

Lines 117 – 120:

```
#ifdef MMS_IO
  UWord8 packed_bits[MAX_PACKED_SIZE];
  Word16 packed_size;
#endif
```

Lines 238 – 241:

```
#ifdef MMS_IO
  UWord8 packed_bits[MAX_PACKED_SIZE];
  Word16 packed_size;
#endif
```

Lines 283 – 310:

```
#ifndef MMS_IO
     serial[0] = tx_type;
     if (tx_type != TX_NO_DATA) {
       serial[1+MAX_SERIAL_SIZE] = mode;
     }
     else {
       serial[1+MAX_SERIAL_SIZE] = -1;
     }

     /* write bitstream to output file */
     if (fwrite (serial, sizeof (Word16), SERIAL_FRAMESIZE, file_serial)
         != SERIAL_FRAMESIZE) {
         fprintf(stderr, "\nerror writing output file: %s\n",
                 strerror(errno));
         exit(-1);
     }
#else
```

```
    packed_size = PackBits(used_mode, mode, tx_type, &serial[1], packed_bits);

    /* write file storage format bitstream to output file */
    if (fwrite (packed_bits, sizeof (UWord8), packed_size, file_serial)
        != packed_size) {
        fprintf(stderr, "\nerror writing output file: %s\n",
                strerror(errno));
        exit(-1);
    }
#endif
```

Changes in file *sp_enc.c*

Lines 41 – 44

```
#ifdef MMS_IO
#include "frame.h"
#include "bitno.tab"
#endif
```

Lines 235 – 330

```
#ifdef MMS_IO

/*************************************************************************
 *
 *  FUNCTION:    PackBits
 *
 *  PURPOSE:     Sorts speech bits according decreasing subjective importance
 *               and packs into octets according to AMR file storage format
 *               as specified in RFC 3267 (Sections 5.1 and 5.3).
 *
 *  DESCRIPTION: Depending on the mode, different numbers of bits are
 *               processed. Details can be found in specification mentioned
 *               above and in file "bitno.tab".
 *
 *************************************************************************/
Word16 PackBits(
    enum Mode used_mode,        /* i : actual AMR mode              */
    enum Mode mode,             /* i : requested AMR (speech) mode */
    enum TXFrameType fr_type,   /* i : frame type                   */
    Word16 bits[],              /* i : serial bits                  */
    UWord8 packed_bits[]        /* o : sorted&packed bits           */
)
{
   Word16 i;
   UWord8 temp;
   UWord8 *pack_ptr;

   temp = 0;
   pack_ptr = (UWord8*)packed_bits;

   /* file storage format can handle only speech frames, AMR SID frames and
NO_DATA frames */
   /* -> force NO_DATA frame */
   if (used_mode < 0 || used_mode > 15 || (used_mode > 8 && used_mode < 15))
   {
      used_mode = 15;
   }

   /* mark empty frames between SID updates as NO_DATA frames */
   if (used_mode == MRDTX && fr_type == TX_NO_DATA)
   {
      used_mode = 15;
   }

   /* insert table of contents (ToC) byte at the beginning of the frame */
   *pack_ptr = toc_byte[used_mode];
```

```
    pack_ptr++;

    /* note that NO_DATA frames (used_mode==15) do not need further processing */
    if (used_mode == 15)
    {
       return 1;
    }

    temp = 0;

    /* sort and pack speech bits */
    for (i = 1; i < unpacked_size[used_mode] + 1; i++)
    {
       if (bits[sort_ptr[used_mode][i-1]] == BIT_1)
       {
          temp++;
       }

       if (i % 8)
       {
          temp <<= 1;
       }
       else
       {
          *pack_ptr = temp;
          pack_ptr++;
          temp = 0;
       }
    }

    /* insert SID type indication and speech mode in case of SID frame */
    if (used_mode == MRDTX)
    {
       if (fr_type == TX_SID_UPDATE)
       {
          temp++;
       }
       temp <<= 3;

       temp += mode & 0x0007;
       temp <<= 1;
    }

    /* insert unused bits (zeros) at the tail of the last byte */
    temp <<= (unused_size[used_mode] - 1);
    *pack_ptr = temp;

    return packed_size[used_mode];
}

#endif
```

Changes in file *sp_enc.h*

Lines 80 – 90

```
#ifdef MMS_IO

Word16 PackBits(
    enum Mode used_mode,        /* i : actual AMR mode             */
    enum Mode mode,             /* i : requested AMR (speech) mode */
    enum TXFrameType fr_type,   /* i : frame type                  */
    Word16 bits[],              /* i : serial bits                 */
    UWord8 packed_bits[]        /* o : sorted&packed bits          */
);

#endif
```

Changes in file *decoder.c*

Lines 53 – 56

```
#ifdef MMS_IO
#define AMR_MAGIC_NUMBER "#!AMR\n"
#define MAX_PACKED_SIZE (MAX_SERIAL_SIZE / 8 + 2)
#endif
```

Lines 114 – 152

```
#ifdef MMS_IO
  UWord8 toc, q, ft;
  Word8 magic[8];
  UWord8 packed_bits[MAX_PACKED_SIZE];
  Word16 packed_size[16] = {12, 13, 15, 17, 19, 20, 26, 31, 5, 0, 0, 0, 0, 0,
0};
#endif

   proc_head ("Decoder");

#ifndef MMS_IO
  /*-----------------------------------------------------------------------*
   * process command line options                                          *
   *-----------------------------------------------------------------------*/
  while (argc > 1) {
      if (strcmp(argv[1], "-rxframetype") == 0)
          rxframetypeMode = 1;
      else break;

      argc--;
      argv++;
  }
#endif

  /*-----------------------------------------------------------------------*
   * check number of arguments                                             *
   *-----------------------------------------------------------------------*/
  if (argc != 3)
  {
    fprintf (stderr,
      " Usage:\n\n"
#ifndef MMS_IO
      "   %s  [-rxframetype] bitstream_file synth_file\n\n"
      " -rxframetype expects the RX frame type in bitstream_file (instead of
TX)\n\n",
#else
      "   %s  bitstream_file synth_file\n\n",
#endif
              progname);
      exit (1);
  }
```

Lines 180 – 190

```
#ifdef MMS_IO
   /* read and verify magic number */
  fread(magic, sizeof(Word8), strlen(AMR_MAGIC_NUMBER), file_serial);
  if (strncmp(magic, AMR_MAGIC_NUMBER, strlen(AMR_MAGIC_NUMBER)))
  {
     fprintf(stderr, "%s%s\n", "Invalid magic number: ", magic);
     fclose(file_serial);
     fclose(file_syn);
     return 1;
   }
#endif
```

Lines 198 – 239

```
  /*-----------------------------------------------------------------------*
   * process serial bitstream frame by frame                               *
   *-----------------------------------------------------------------------*/
  frame = 0;

#ifndef MMS_IO
  while (fread (serial, sizeof (Word16), SERIAL_FRAMESIZE, file_serial)
         == SERIAL_FRAMESIZE)
  {
    ++frame;
    if ( (frame%50) == 0) {
       fprintf (stderr, "\rframe=%d  ", frame);
    }

    /* get frame type and mode information from frame */
    if (rxframetypeMode) {
        rx_type = (enum RXFrameType)serial[0];
    } else {
        tx_type = (enum TXFrameType)serial[0];
        rx_type = tx_to_rx (tx_type);
    }
    mode = (enum Mode) serial[1+MAX_SERIAL_SIZE];

#else

  while (fread (&toc, sizeof(UWord8), 1, file_serial) == 1)
  {
    /* read rest of the frame based on ToC byte */
    q  = (toc >> 2) & 0x01;
    ft = (toc >> 3) & 0x0F;
    fread (packed_bits, sizeof(UWord8), packed_size[ft], file_serial);

    rx_type = UnpackBits(q, ft, packed_bits, &mode, &serial[1]);

#endif

    ++frame;
    if ( (frame%50) == 0) {
       fprintf (stderr, "\rframe=%d  ", frame);
    }

    if (rx_type == RX_NO_DATA) {
      mode = speech_decoder_state->prev_mode;
    }
    else {
      speech_decoder_state->prev_mode = mode;
    }
```

Changes in file *sp_dec.c*

Lines 42 – 44

```
#ifdef MMS_IO
#include "bitno.tab"
#endif
```

Lines 220 – 298

```
#ifdef MMS_IO

/*
*************************************************************************
*
*  Function    : UnpackBits
*  Purpose     : Unpack and re-arrange bits from file storage format to the
*                format required by speech decoder.
```

```
*
*****************************************************************************
*/
enum RXFrameType UnpackBits (
    Word8  q,                /* i : Q-bit (i.e. BFI)         */
  Word16 ft,                 /* i : frame type (i.e. mode)   */
    UWord8 packed_bits[],    /* i : sorted & packed bits     */
  enum Mode *mode,           /* o : mode information         */
    Word16 bits[]            /* o : serial bits              */
)
{
  Word16 i, sid_type;
  UWord8 *pack_ptr, temp;

  pack_ptr = (UWord8*)packed_bits;

  /* real NO_DATA frame or unspecified frame type */
  if (ft == 15 || (ft > 8 && ft < 15))
  {
    *mode = (enum Mode)-1;
    return RX_NO_DATA;
  }

  temp = *pack_ptr;
  pack_ptr++;

  for (i = 1; i < unpacked_size[ft] + 1; i++)
  {
    if (temp & 0x80) bits[sort_ptr[ft][i-1]] = BIT_1;
    else             bits[sort_ptr[ft][i-1]] = BIT_0;

    if (i % 8)
    {
      temp <<= 1;
    }
    else
    {
      temp = *pack_ptr;
      pack_ptr++;
    }
  }

  /* SID frame */
  if (ft == MRDTX)
  {
    if (temp & 0x80) sid_type = 1;
    else             sid_type = 0;

    *mode = (enum Mode)((temp >> 4) & 0x07);

    if (q)
    {
      if (sid_type) return  RX_SID_UPDATE;
      else      return RX_SID_FIRST;
    }
    else
    {
      return RX_SID_BAD;
    }
  }
  /* speech frame */
  else
  {
    *mode = (enum Mode)ft;

    if (q) return RX_SPEECH_GOOD;
    else return RX_SPEECH_BAD;
  }
}

#endif
```

Changes in file *sp_dec.h*

Lines 79 – 88

```
#ifdef MMS_IO

enum RXFrameType UnpackBits (
    Word8  q,                /* i : Q-bit (i.e. BFI)        */
   Word16 ft,              /* i : frame type (i.e. mode)  */
    UWord8 packed_bits[],  /* i : sorted & packed bits    */
   enum Mode *mode,        /* o : mode information        */
    Word16 bits[]           /* o : serial bits             */
);
#endif
```

Changes in file *bitno.tab*

Lines 159 – 337

```
#ifdef MMS_IO

/* table of contents byte for each mode index */
static UWord8 toc_byte[16] = {0x04, 0x0C, 0x14, 0x1C, 0x24, 0x2C, 0x34, 0x3C,
                              0x44, 0x4C, 0x54, 0x5C, 0x64, 0x6C, 0x74, 0x7C};

/* number of speech bits for all modes */
static Word16 unpacked_size[16] = {95, 103, 118, 134, 148, 159, 204, 244,
                                   35,   0,   0,   0,   0,   0,   0,   0};

/* size of packed frame for each mode */
static Word16 packed_size[16] = {13, 14, 16, 18, 20, 21, 27, 32,
                                 6,  0,  0,  0,  0,  0,  0,  1};

/* number of unused speech bits in packed format for each mode */
static Word16 unused_size[16] = {1, 1, 2, 2, 4, 1, 4, 4, 1, 0, 0, 0, 0, 0, 0,
0};

/* sorting tables for all modes */

static Word16 sort_475[95] = {
    0,  1,  2,  3,  4,  5,  6,  7,  8,  9,
   10, 11, 12, 13, 14, 15, 23, 24, 25, 26,
   27, 28, 48, 49, 61, 62, 82, 83, 47, 46,
   45, 44, 81, 80, 79, 78, 17, 18, 20, 22,
   77, 76, 75, 74, 29, 30, 43, 42, 41, 40,
   38, 39, 16, 19, 21, 50, 51, 59, 60, 63,
   64, 72, 73, 84, 85, 93, 94, 32, 33, 35,
   36, 53, 54, 56, 57, 66, 67, 69, 70, 87,
   88, 90, 91, 34, 55, 68, 89, 37, 58, 71,
   92, 31, 52, 65, 86
};

static Word16 sort_515[103] = {
    7,  6,  5,   4,   3,   2,  1,  0, 15, 14,
   13, 12, 11,  10,   9,   8, 23, 24, 25, 26,
   27, 46, 65,  84,  45,  44, 43, 64, 63, 62,
   83, 82, 81, 102, 101, 100, 42, 61, 80, 99,
   28, 47, 66,  85,  18,  41, 60, 79, 98, 29,
   48, 67, 17,  20,  22,  40, 59, 78, 97, 21,
   30, 49, 68,  86,  19,  16, 87, 39, 38, 58,
   57, 77, 35,  54,  73,  92, 76, 96, 95, 36,
   55, 74, 93,  32,  51,  33, 52, 70, 71, 89,
   90, 31, 50,  69,  88,  37, 56, 75, 94, 34,
   53, 72, 91
};

static Word16 sort_59[118] = {
```

```
    0,   1,   4,   5,   3,   6,   7,   2,  13,  15,
    8,   9,  11,  12,  14,  10,  16,  28,  74,  29,
   75,  27,  73,  26,  72,  30,  76,  51,  97,  50,
   71,  96, 117,  31,  77,  52,  98,  49,  70,  95,
  116,  53,  99,  32,  78,  33,  79,  48,  69,  94,
  115,  47,  68,  93, 114,  46,  67,  92, 113,  19,
   21,  23,  22,  18,  17,  20,  24, 111,  43,  89,
  110,  64,  65,  44,  90,  25,  45,  66,  91, 112,
   54, 100,  40,  61,  86, 107,  39,  60,  85, 106,
   36,  57,  82, 103,  35,  56,  81, 102,  34,  55,
   80, 101,  42,  63,  88, 109,  41,  62,  87, 108,
   38,  59,  84, 105,  37,  58,  83, 104
};

static Word16 sort_67[134] = {
    0,   1,   4,   3,   5,   6,  13,   7,   2,   8,
    9,  11,  15,  12,  14,  10,  28,  82,  29,  83,
   27,  81,  26,  80,  30,  84,  16,  55, 109,  56,
  110,  31,  85,  57, 111,  48,  73, 102, 127,  32,
   86,  51,  76, 105, 130,  52,  77, 106, 131,  58,
  112,  33,  87,  19,  23,  53,  78, 107, 132,  21,
   22,  18,  17,  20,  24,  25,  50,  75, 104, 129,
   47,  72, 101, 126,  54,  79, 108, 133,  46,  71,
  100, 125, 128, 103,  74,  49,  45,  70,  99, 124,
   42,  67,  96, 121,  39,  64,  93, 118,  38,  63,
   92, 117,  35,  60,  89, 114,  34,  59,  88, 113,
   44,  69,  98, 123,  43,  68,  97, 122,  41,  66,
   95, 120,  40,  65,  94, 119,  37,  62,  91, 116,
   36,  61,  90, 115
};

static Word16 sort_74[148] = {
    0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
   10,  11,  12,  13,  14,  15,  16,  26,  87,  27,
   88,  28,  89,  29,  90,  30,  91,  51,  80, 112,
  141,  52,  81, 113, 142,  54,  83, 115, 144,  55,
   84, 116, 145,  58, 119,  59, 120,  21,  22,  23,
   17,  18,  19,  31,  60,  92, 121,  56,  85, 117,
  146,  20,  24,  25,  50,  79, 111, 140,  57,  86,
  118, 147,  49,  78, 110, 139,  48,  77,  53,  82,
  114, 143, 109, 138,  47,  76, 108, 137,  32,  33,
   61,  62,  93,  94, 122, 123,  41,  42,  43,  44,
   45,  46,  70,  71,  72,  73,  74,  75, 102, 103,
  104, 105, 106, 107, 131, 132, 133, 134, 135, 136,
   34,  63,  95, 124,  35,  64,  96, 125,  36,  65,
   97, 126,  37,  66,  98, 127,  38,  67,  99, 128,
   39,  68, 100, 129,  40,  69, 101, 130
};

static Word16 sort_795[159] = {
    8,   7,   6,   5,   4,   3,   2,  14,  16,   9,
   10,  12,  13,  15,  11,  17,  20,  22,  24,  23,
   19,  18,  21,  56,  88, 122, 154,  57,  89, 123,
  155,  58,  90, 124, 156,  52,  84, 118, 150,  53,
   85, 119, 151,  27,  93,  28,  94,  29,  95,  30,
   96,  31,  97,  61, 127,  62, 128,  63, 129,  59,
   91, 125, 157,  32,  98,  64, 130,   1,   0,  25,
   26,  33,  99,  34, 100,  65, 131,  66, 132,  54,
   86, 120, 152,  60,  92, 126, 158,  55,  87, 121,
  153, 117, 116, 115,  46,  78, 112, 144,  43,  75,
  109, 141,  40,  72, 106, 138,  36,  68, 102, 134,
  114, 149, 148, 147, 146,  83,  82,  81,  80,  51,
   50,  49,  48,  47,  45,  44,  42,  39,  35,  79,
   77,  76,  74,  71,  67, 113, 111, 110, 108, 105,
  101, 145, 143, 142, 140, 137, 133,  41,  73, 107,
  139,  37,  69, 103, 135,  38,  70, 104, 136
};

static Word16 sort_102[204] = {
    7,   6,   5,   4,   3,   2,   1,   0,  16,  15,
   14,  13,  12,  11,  10,   9,   8,  26,  27,  28,
```

```
   29,  30,  31, 115, 116, 117, 118, 119, 120,  72,
   73, 161, 162,  65,  68,  69, 108, 111, 112, 154,
  157, 158, 197, 200, 201,  32,  33, 121, 122,  74,
   75, 163, 164,  66, 109, 155, 198,  19,  23,  21,
   22,  18,  17,  20,  24,  25,  37,  36,  35,  34,
   80,  79,  78,  77, 126, 125, 124, 123, 169, 168,
  167, 166,  70,  67,  71, 113, 110, 114, 159, 156,
  160, 202, 199, 203,  76, 165,  81,  82,  92,  91,
   93,  83,  95,  85,  84,  94, 101, 102,  96, 104,
   86, 103,  87,  97, 127, 128, 138, 137, 139, 129,
  141, 131, 130, 140, 147, 148, 142, 150, 132, 149,
  133, 143, 170, 171, 181, 180, 182, 172, 184, 174,
  173, 183, 190, 191, 185, 193, 175, 192, 176, 186,
   38,  39,  49,  48,  50,  40,  52,  42,  41,  51,
   58,  59,  53,  61,  43,  60,  44,  54, 194, 179,
  189, 196, 177, 195, 178, 187, 188, 151, 136, 146,
  153, 134, 152, 135, 144, 145, 105,  90, 100, 107,
   88, 106,  89,  98,  99,  62,  47,  57,  64,  45,
   63,  46,  55,  56
};

static Word16 sort_122[244] = {
    0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
   10,  11,  12,  13,  14,  23,  15,  16,  17,  18,
   19,  20,  21,  22,  24,  25,  26,  27,  28,  38,
  141,  39, 142,  40, 143,  41, 144,  42, 145,  43,
  146,  44, 147,  45, 148,  46, 149,  47,  97, 150,
  200,  48,  98, 151, 201,  49,  99, 152, 202,  86,
  136, 189, 239,  87, 137, 190, 240,  88, 138, 191,
  241,  91, 194,  92, 195,  93, 196,  94, 197,  95,
  198,  29,  30,  31,  32,  33,  34,  35,  50, 100,
  153, 203,  89, 139, 192, 242,  51, 101, 154, 204,
   55, 105, 158, 208,  90, 140, 193, 243,  59, 109,
  162, 212,  63, 113, 166, 216,  67, 117, 170, 220,
   36,  37,  54,  53,  52,  58,  57,  56,  62,  61,
   60,  66,  65,  64,  70,  69,  68, 104, 103, 102,
  108, 107, 106, 112, 111, 110, 116, 115, 114, 120,
  119, 118, 157, 156, 155, 161, 160, 159, 165, 164,
  163, 169, 168, 167, 173, 172, 171, 207, 206, 205,
  211, 210, 209, 215, 214, 213, 219, 218, 217, 223,
  222, 221,  73,  72,  71,  76,  75,  74,  79,  78,
   77,  82,  81,  80,  85,  84,  83, 123, 122, 121,
  126, 125, 124, 129, 128, 127, 132, 131, 130, 135,
  134, 133, 176, 175, 174, 179, 178, 177, 182, 181,
  180, 185, 184, 183, 188, 187, 186, 226, 225, 224,
  229, 228, 227, 232, 231, 230, 235, 234, 233, 238,
  237, 236,  96, 199
};

static Word16 sort_SID[35] = {
    0,  1,  2,  3,  4,  5,  6,  7,  8,  9,
   10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
   20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
   30, 31, 32, 33, 34
};

/* pointer table for bit sorting tables */
static Word16 *sort_ptr[16] = {sort_475, sort_515, sort_59, sort_67, sort_74,
sort_795, sort_102, sort_122,
                               sort_SID,     NULL,    NULL,     NULL,     NULL,
NULL,     NULL,       NULL};

#endif
```