Technical Specification Group Services and System Aspects ***TSGS#19(03)0089***
Meeting #19, Birmingham, UK, 17 - 20 March 2003

Source: **TSG-SA WG4**

Title: CRs to TS 26.173 Harmonization of 3GPP TS 26.173 and ITU-T G.722.2 C-codes, and Correction for handling of RX_NO_DATA frames (Release 5)

Document for: **Approval**

Agenda Item: **7.4.3**

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

Spec	CR	Rev	Phase	Subject	Cat	Vers	WG	Meeting	S4 doc
26.173	015	2	Rel-5	Harmonization of 3GPP TS 26.173 and ITU-T G.722.2 C-codes	F	5.5.0	S4	TSG-SA WG4#25	S4-030027
26.173	016		Rel-5	Correction for handling of RX_NO_DATA frames	F	5.5.0	S4	TSG-SA WG4#25bis	S4-030143

3GPP TSG-SA4 Meeting #25
San Francisco, USA, 20-24 January 2003

Tdoc ⌘S4-030027

CR-Form-v7

CHANGE REQUEST

⌘ **26.173** **CR** **015** ⌘**rev** **2** ⌘ Current version: **5.5.0** ⌘

*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:	⌘	Harmonization of 3GPP TS26.173 and ITU-T G.722.2 C-codes
Source:	⌘	TSG SA WG4
Work item code:	⌘	AMRWB
Date:	⌘	18/02/2003
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 represented in this document harmonise the I/O of 3GPP and ITU-T C-codes of AMR-WB codec. Using the very same reference code in both standardisation bodies makes the maintenance of the C-code much easier and also guarantees full interoperability between implementations based on 3GPP and ITU-T C-codes.
Summary of change:	⌘	New option of ITU-T bit stream format added.
Consequences if not approved:	⌘	Different reference C-codes for the same algorithm exists. Continuing maintenance of two different C-codes may even result in non-interoperable implementations.

Clauses affected:	⌘	26.173-550.doc, coder.c, decoder.c bits.c and bits.h

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

Other comments: ⌘

Changes to the specification document (26.173-550.doc):

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 available formats.

NOTE ON DEFAULT 3GPP AND ITU BITSTREAM FORMATS:
ITU stream format gives very limited possibilities to distinguish NO_DATA and SID_FIRST frame types at the beginning of a stream. In some very limited cases for which some instance between encoder and decoder cuts of the first hangover period frames (e.g. handovers, editing of the stream), the output of the decoder is different depending on the stream format, ITU or default 3GPP.

Default 3GPP format:

This is the default format used in 3GPP. This format shall be used when the codec is tested against the test vectors.

TYPE_OF_FRAME_TYPE	FRAME_TYPE	MODE	B1	B2	…	Bnn

Each box corresponds to one `Word16` value in the bitstream file, for a total of 3+nn words or 6+2nn bytes per frame, where nn is the number of encoded bits in the frame. Each encoded bit is represented as follows: Bit 0 = 0xff81, Bit 1 = 0x007f. The fields have the following meaning:

`TYPE_OF_FRAME_TYPE` transmit frame type, which is one of
`TX_TYPE (0x6b21)`
`RX_TYPE (0x6b20)`

`If TYPE_OF_FRAME_TYPE is TX_TYPE,`

`FRAME_TYPE` transmit frame type, which is one of
`TX_SPEECH (0x0000)`
`TX_SID_FIRST (0x0001)`
`TX_SID_UPDATE (0x0002)`
`TX_NO_DATA (0x0003)`

`If TYPE_OF_FRAME_TYPE is RX_TYPE,`

`FRAME_TYPE` transmit frame type, which is one of
`RX_SPEECH_GOOD (0x0000)`
`RX_SPEECH_PROBABLY_DEGRADED (0x0001)`
`RX_SPEECH_LOST (0x0002)`
`RX_SPEECH_BAD (0x0003)`
`RX_SID_FIRST (0x0004)`
`RX_SID_UPDATE (0x0005)`
`RX_SID_BAD (0x0006)`
`RX_NO_DATA (0x0007)`

`B0…B2nn` speech encoder parameter bits (i.e. the bitstream itself). Each `Bx` either has the value `0x0081 (for bit 0)` or `0x007F (for bit 1)`.

`MODE_INFO` encoding mode information, which is one of
`6.60 kbit/s mode (0x0000)`
`8.85 kbit/s mode (0x0001)`
`12.65 kbit/s mode (0x0002)`
`14.25 kbit/s mode (0x0003)`
`15.85 kbit/s mode (0x0004)`
`18.25 kbit/s mode (0x0005)`
`19.85 kbit/s mode (0x0006)`
`23.05 kbit/s mode (0x0007)`
`23.85 kbit/s mode (0x0008)`

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

ITU format (activated with command line parameter -itu)

SYNC_WORD	DATA_LENGTH	B1	B2	…	Bnn

Each box corresponds to one `Word16` value in the bitstream file, for a total of 2+nn words or 4+2nn bytes per frame, where nn is the number of encoded bits in the frame. Each encoded bit is represented as follows: Bit 0 = 0x007f, Bit 1 = 0x0081. The fields have the following meaning:

SYNC_WORD — Word to ensure correct frame synchronization between the encoder and the decoder. It is also used to indicate the occurrences of bad frames.

```
In the encoder output:   (0x6b21)
In the decoder input:    Good frames (0x6b21)
                         Bad frames  (0x6b20)
```

DATA_LENGTH — Length of the speech data. Codec mode and frame type is extracted in the decoder using this parameter:

DATA_LENGTH	PREVIOUS FRAME	CODEC MODE	FRAMETYPE
0	RX_SPEECH_GOOD/ RX_SPEECH_BAD	DTX	RX_SID_FIRST
0	OTHER THAN RX_SPEECH_GOOD/ RX_SPEECH_BAD	DTX	RX_NO_DATA
35	-	DTX	RX_SID_UPDATE/ RX_SID_BAD
132	-	6.60 kbit/s	RX_SPEECH_GOOD/ RX_SPEECH_BAD
177	-	8.85 kbit/s	RX_SPEECH_GOOD/ RX_SPEECH_BAD
253	-	12.65 kbit/s	RX_SPEECH_GOOD/ RX_SPEECH_BAD
285	-	14.25 kbit/s	RX_SPEECH_GOOD/ RX_SPEECH_BAD
317	-	15.85 kbit/s	RX_SPEECH_GOOD/ RX_SPEECH_BAD
365	-	18.25 kbit/s	RX_SPEECH_GOOD/ RX_SPEECH_BAD
397	-	19.85 kbit/s	RX_SPEECH_GOOD/ RX_SPEECH_BAD
461	-	23.05 kbit/s	RX_SPEECH_GOOD/ RX_SPEECH_BAD
477	-	23.85 kbit/s	RX_SPEECH_GOOD/ RX_SPEECH_BAD

Changes to the C-code:

1. How the code is changed in the file *cnst.h*

Line 7:

```
#define CODEC_VERSION "5.56.0"
```

2. How the code is changed in the file *bits.h*

Lines 22-29:

```
#define NB_BITS_MAX   NBBITS_24k

#define BIT_0     (Word16)-127
#define BIT_1     (Word16)127
#define BIT_0_ITU (Word16)0x007F
#define BIT_1_ITU (Word16)0x0081

#define SIZE_MAX  (3+NB_BITS_MAX)          /* serial size max */
```

Lines 50-56:

```
typedef struct
{
    Word16 prev_ft;
    Word16 prev_mode;
} RX_State;

Word16 Init_write_serial(TX_State ** st);
Word16 Close_write_serial(TX_State *st);
void Reset_write_serial(TX_State * st);
Word16 Init_read_serial(RX_State ** st);
Word16 Close_read_serial(RX_State *st);
void Reset_read_serial(RX_State * st);
void Write_serial(FILE * fp, Word16 prms[], Word16 coding_mode, Word16 mode, TX_State *st, Word16
bitstreamformat);
Word16 Read_serial(FILE * fp, Word16 prms[], Word16 * frame_type, Word16 * mode, RX_State *st,
Word16 bitstreamformat);
```

3. How the code is changed in the file *bits.c*

Lines 58-159:

```
void Write_serial(FILE * fp, Word16 prms[], Word16 coding_mode, Word16 mode, TX_State *st, Word16
bitstreamformat)
{
   Word16 i, frame_type;
   Word16 stream[SIZE_MAX];

   if (coding_mode == MRDTX)
   {

      st->sid_update_counter--;

      if (st->prev_ft == TX_SPEECH)
      {
         frame_type = TX_SID_FIRST;
         st->sid_update_counter = 3;
      } else
      {
         if ((st->sid_handover_debt > 0) &&
         (st->sid_update_counter > 2))
         {
         /* ensure extra updates are  properly delayed after a possible SID_FIRST */
            frame_type = TX_SID_UPDATE;
            st->sid_handover_debt--;
         } else
         {
            if (st->sid_update_counter == 0)
            {
               frame_type = TX_SID_UPDATE;
               st->sid_update_counter = 8;
            } else
            {
               frame_type = TX_NO_DATA;
            }
```

```
         }
      }
   } else
   {
      st->sid_update_counter = 8;
      frame_type = TX_SPEECH;
   }
   st->prev_ft = frame_type;


   if(bitstreamformat == 0)          /* default file format */
   {
      stream[0] = TX_FRAME_TYPE;
      stream[1] = frame_type;
      stream[2] = mode;
      for (i = 0; i < nb_of_bits[coding_mode]; i++)
      {
         stream[3 + i] = prms[i];
      }

      fwrite(stream, sizeof(Word16), 3 + nb_of_bits[coding_mode], fp);

   } else                        /* ITU file format */
   {
      stream[0] = 0x6b21;

      if(frame_type != TX_NO_DATA && frame_type != TX_SID_FIRST)
      {
         stream[1]=nb_of_bits[coding_mode];
         for (i = 0; i < nb_of_bits[coding_mode]; i++)
         {
            if(prms[i] == BIT_0){
               stream[2 + i] = BIT_0_ITU;
            }
            else{
               stream[2 + i] = BIT_1_ITU;
            }

         }
         fwrite(stream, sizeof(Word16), 2 + nb_of_bits[coding_mode], fp);
      } else
      {
         stream[1] = 0;
         fwrite(stream, sizeof(Word16), 2, fp);
      }
   }
   return;
}


/*-----------------------------------------------------*
 * Read_serial -> read serial stream into a file       *
 *-----------------------------------------------------*/
Word16 Init_read_serial(RX_State ** st)
{
   RX_State *s;

   /* allocate memory */
   test();
   if ((s = (RX_State *) malloc(sizeof(RX_State))) == NULL)
   {
      fprintf(stderr, "read_serial_init: can not malloc state structure\n");
      return -1;
   }
   Reset_read_serial(s);
   *st = s;

   return 0;
}

Word16 Close_read_serial(RX_State *st)
{
   /* allocate memory */
   test();
   if (st != NULL)
   {
      free(st);
      st = NULL;
      return 0;
   }
   return 1;
}

void Reset_read_serial(RX_State * st)
{
   st->prev_ft = RX_SPEECH_GOOD;
   st->prev_mode = 0;
}

Word16 Read_serial(FILE * fp, Word16 prms[], Word16 * frame_type, Word16 * mode, RX_State *st,
Word16 bitstreamformat)
```

```
{
   Word16 n, n1, type_of_frame_type, coding_mode, datalen, i;

   if(bitstreamformat == 0)          /* default file format */
   {
      n = (Word16) fread(&type_of_frame_type, sizeof(Word16), 1, fp);
      n = (Word16) (n + fread(frame_type, sizeof(Word16), 1, fp));
      n = (Word16) (n + fread(mode, sizeof(Word16), 1, fp));
      coding_mode = *mode;
      if(*mode < 0 || *mode > NUM_OF_MODES-1)
      {
         fprintf(stderr, "Invalid mode received: %d (check file format).\n", *mode);
         exit(-1);
      }
      if (n == 3)
      {
         if (type_of_frame_type == TX_FRAME_TYPE)
         {
            switch (*frame_type)
            {
               case TX_SPEECH:
                  *frame_type = RX_SPEECH_GOOD;
                   break;
               case TX_SID_FIRST:
                  *frame_type = RX_SID_FIRST;
                  break;
               case TX_SID_UPDATE:
                  *frame_type = RX_SID_UPDATE;
                  break;
               case TX_NO_DATA:
                  *frame_type = RX_NO_DATA;
                  break;
            }
         } else if (type_of_frame_type != RX_FRAME_TYPE)
         {
            fprintf(stderr, "Wrong type of frame type:%d.\n", type_of_frame_type);
         }

         if ((*frame_type == RX_SID_FIRST) | (*frame_type == RX_SID_UPDATE) | (*frame_type ==
RX_NO_DATA) | (*frame_type == RX_SID_BAD))
         {
            coding_mode = MRDTX;
         }
         n = (Word16) fread(prms, sizeof(Word16), nb_of_bits[coding_mode], fp);
         if (n != nb_of_bits[coding_mode])
            n = 0;
      }
      return (n);
   } else                        /* ITU file format */
   {
      n = (Word16) fread(&type_of_frame_type, sizeof(Word16), 1, fp);
      n = (Word16)(n+fread(&datalen, sizeof(Word16), 1, fp));

      if(n == 2)
      {
         if(datalen == 0)                    /* RX_NO_DATA frame type */
         {
            if(st->prev_ft == RX_SPEECH_GOOD || st->prev_ft == RX_SPEECH_BAD)
            {
               *frame_type = RX_SID_FIRST;
            } else
            {
               *frame_type = RX_NO_DATA;
            }
         *mode = st->prev_mode;
      }
      else{

         coding_mode = -1;
         for(i=NUM_OF_MODES-1; i>=0; i--)
         {
            if(datalen == nb_of_bits[i])
            {
               coding_mode = i;
            }
         }
         if(coding_mode == -1)
         {
            fprintf(stderr, "\n\n ERROR: Invalid number of data bits received [%d]\n\n", datalen);
            exit(-1);
         }
```

```
         if(coding_mode == NUM_OF_MODES-1)    /* DTX frame type */
         {
            if(type_of_frame_type == 0x6b20) /* bad SID frame */
            {
               *frame_type = RX_SID_BAD;
            } else                              /* correct SID frame */
            {
               *frame_type = RX_SID_UPDATE;
            }
            *mode = st->prev_mode;
            } else
            {
               if(type_of_frame_type == 0x6b20)
               {
                  *frame_type = RX_SPEECH_BAD;
               } else
               {
                  *frame_type = RX_SPEECH_GOOD;
               }
               *mode = coding_mode;
            }
         }
      }
      n1 =fread(prms, sizeof(Word16), datalen, fp);
      n += n1;
      for(i=0; i<n1; i++){
         if(prms[i] <= BIT_0_ITU) prms[i] = BIT_0;
         else                     prms[i] = BIT_1;
      }
      st->prev_mode = *mode;
      st->prev_ft = *frame_type;
      return(n);
   }
```

4. How the code is changed in the file *coder.c*

Lines 22-45:

```
/*-----------------------------------------------------------------*
 * CODER.C                                                         *
 * ~~~~~~~~~~                                                      *
 * Main program of the AMR WB ACELP wideband coder.                *
 *                                                                 *
 *    Usage : coder (-dtx) (-itu) mode speech_file  bitstream_file *
 *                                                                 *
 *    Format for speech_file:                                      *
 *      Speech is read from a binary file of 16 bits data.         *
 *                                                                 *
 *    Format for bitstream_file (default):                         *
 *                                                                 *
 *        1 word (2-byte) for the type of frame type               *
 *          (TX_FRAME_TYPE or RX_FRAME_TYPE)                       *
 *        1 word (2-byte) for the frame type                       *
 *          (see dtx.h for possible values)                        *
 *        1 word (2-byte) for the mode indication                  *
 *          (see bits.h for possible values)                       *
 *        N words (2-byte) containning N bits.                     *
 *          Bit 0 = 0x0000ff81 and Bit 1 = 0x0001007f                      *
 *                                                                 *
 *    if option -itu defined:                                      *
 *        1 word (2-byte) for sync word (0x6b21)                   *
 *        1 word (2-byte) frame length N                           *
 *        N words (2-byte) containing N 'soft' bits                *
 *              (bit 0 = 0x007f, bit 1 = 0x0081)                   *
 *                                                                 *
 *    mode = 0..8 (bit rate = 6.60 to 23.85 k)                     *
 *                                                                 *
 *    -dtx if DTX is ON                                            *
 *-----------------------------------------------------------------*/
```

Lines 56-103:

```
   Word16 coding_mode = 0, nb_bits, allow_dtx, mode_file, mode = 0, i;
   Word16 bitstreamformat;
   Word16 reset_flag;
   long frame;

   void *st;
   TX_State *tx_state;
```

```
   fprintf(stderr, "\n");
   fprintf(stderr, "
================================================================================================
\n");
   fprintf(stderr, "3GPP AMR Wideband Codec 3GPP TS26.190 / ITU-T G.722.2, February 15 October 10,
2002. Version %s.\n", CODEC_VERSION);
   fprintf(stderr, "
================================================================================================
\n");
   fprintf(stderr, "\n");

   /*-----------------------------------------------------------------------*
    * Open speech file and result file (output serial bit stream)           *
    *-----------------------------------------------------------------------*/

   if ((argc < 4) || (argc > 6))
   {
      fprintf(stderr, "Usage : coder  (-dtx) (-itu) mode speech_file  bitstream_file\n");
      fprintf(stderr, "\n");
      fprintf(stderr, "Format for speech_file:\n");
      fprintf(stderr, "  Speech is read form a binary file of 16 bits data.\n");
      fprintf(stderr, "\n");
      fprintf(stderr, "Format for bitstream_file (default):\n");
      fprintf(stderr, "  One word (2-byte) to indicate type of frame type.\n");
      fprintf(stderr, "  One word (2-byte) to indicate frame type.\n");
      fprintf(stderr, "  One word (2-byte) to indicate mode.\n");
      fprintf(stderr, "  N words (2-byte) containing N bits (bit 0 = 0xff81, bit 1 = 0x007f).\n");
      fprintf(stderr, "\n");
      fprintf(stderr, "  if option -itu defined:\n");
      fprintf(stderr, "  One word (2-byte) for sync word (0x6b21)\n");
      fprintf(stderr, "  One word (2-byte) for frame length N.\n");
      fprintf(stderr, "  N words (2-byte) containing N bits (bit 0 = 0x007f, bit 1 = 0x0081).\n");
      fprintf(stderr, "\n");
      fprintf(stderr, "mode: 0 to 8 (9 bits rates) or\n");
      fprintf(stderr, "      -modefile filename\n");
      fprintf(stderr, " ================================================================\n");
      fprintf(stderr, " mode   :  (0)  (1)   (2)   (3)   (4)   (5)   (6)   (7)   (8)     \n");
      fprintf(stderr, " bitrate: 6.60 8.85 12.65 14.25 15.85 18.25 19.85 23.05 23.85 kbit/s\n");
      fprintf(stderr, " ================================================================\n");
      fprintf(stderr, "\n");
      fprintf(stderr, "-dtx if DTX is ON, default is OFF\n");
      fprintf(stderr, "\n");
      exit(0);
   }

   allow_dtx = 0;
   if (strcmp(argv[1], "-dtx") == 0)
   {
      allow_dtx = 1;
      argv++;
   }

   bitstreamformat = 0;
   if (strcmp(argv[1], "-itu") == 0)
   {
      bitstreamformat = 1;
      argv++;
      fprintf(stderr, "Input bitstream format: ITU\n");
   } else
   {
      fprintf(stderr, "Input bitstream format: Default\n");
   }

   mode_file = 0;
```

Line 194:

```
    Write_serial(f_serial, prms, coding_mode, mode, tx_state, bitstreamformat);
```

5. How the code is changed in the file *decoder.c*

Lines 20-40:

```
/*-----------------------------------------------------------------*
 * DECODER.C                                                       *
 * ~~~~~~~~~                                                       *
 * Main program of the AMR WB ACELP wideband decoder.              *
 *                                                                 *
 *    Usage : decoder (-itu) bitstream_file synth_file bit_rate    *
 *                                                                 *
 *    Format for bitstream_file (default):                         *
 *                                                                 *
 *        1 word (2-byte) for the type of frame type               *
 *          (TX_FRAME_TYPE or RX_FRAME_TYPE)                       *
 *        1 word (2-byte) for the frame type                       *
 *          (see dtx.h for possible values)                        *
 *        1 word (2-byte) for the mode indication                  *
```

```
 *          (see bits.h for possible values)                     *
 *        N words (2-byte) containning N bits.                   *
 *          Bit 0 = 0x0000ff81 and Bit 1 = 0x0001007f            *
 *                                                               *
 *    if option -itu defined:                                    *
 *        1 word (2-byte) for sync word (0x6b21)                 *
 *        1 word (2-byte) frame length N                         *
 *        N words (2-byte) containing N 'soft' bits              *
 *              (bit 0 = 0x007f, bit 1 = 0x0081)                 *
 *                                                               *
 *    Format for synth_file:                                     *
 *      Synthesis is written to a binary file of 16 bits data.   *
 *                                                               *
 *---------------------------------------------------------------*/
```

Lines 55-116:

```
    long frame;

    Word16 bitstreamformat;
    RX_State *rx_state;

    void *st;

    fprintf(stderr, "\n");
    fprintf(stderr, "
================================================================================================
\n");
    fprintf(stderr, "3GPP AMR Wideband Codec 3GPP TS26.190 / ITU-T G.722.2, February 15 October 10,
2002. Version %s.\n", CODEC_VERSION);
    fprintf(stderr, "
================================================================================================
\n");
    fprintf(stderr, "\n");


    /*-----------------------------------------------------------------*
     *           Read passed arguments and open in/out files           *
     *-----------------------------------------------------------------*/

    if (argc != 3 && argc != 4)
    {
       fprintf(stderr, "Usage : decoder  (-itu) bitstream_file  synth_file\n");
       fprintf(stderr, "\n");
       fprintf(stderr, "Format for bitstream_file: (default)\n");
       fprintf(stderr, "  One word (2-byte) to indicate type of frame type.\n");
       fprintf(stderr, "  One word (2-byte) to indicate frame type.\n");
       fprintf(stderr, "  One word (2-byte) to indicate mode.\n");
       fprintf(stderr, "  N words (2-byte) containing N bits (bit 0 = 0xff81, bit 1 = 0x007f).\n");
       fprintf(stderr, "\n");
       fprintf(stderr, "  if option -itu defined:\n");
       fprintf(stderr, "  One word (2-byte) for sync word (good frames: 0x6b21, bad frames:
0x6b20)\n");
       fprintf(stderr, "  One word (2-byte) for frame length N.\n");
       fprintf(stderr, "  N words (2-byte) containing N bits (bit 0 = 0x007f, bit 1 = 0x0081).\n");
       fprintf(stderr, "\n");
       fprintf(stderr, "Format for synth_file:\n");
       fprintf(stderr, "  Synthesis is written to a binary file of 16 bits data.\n");
       fprintf(stderr, "\n");
       exit(0);
    }


    bitstreamformat = 0;
    if (strcmp(argv[1], "-itu") == 0)
    {
       bitstreamformat = 1;
       argv++;
       fprintf(stderr, "Input bitstream format: ITU\n");
    } else
    {
       fprintf(stderr, "Input bitstream format: Default\n");
    }


    /* Open file for synthesis and packed serial stream */

    if ((f_serial = fopen(argv[1], "rb")) == NULL)
    {
       fprintf(stderr, "Input file '%s' does not exist !!\n", argv[1]);
       exit(0);
    } else
       fprintf(stderr, "Input bitstream file:   %s\n", argv[1]);

       if ((f_synth = fopen(argv[2], "wb")) == NULL)
       {
          fprintf(stderr, "Cannot open file '%s' !!\n", argv[2]);
          exit(0);
       } else
          fprintf(stderr, "Synthesis speech file:  %s\n", argv[2]);
     /*-----------------------------------------------------------------*
```

```
      *           Initialization of decoder                         *
      *-------------------------------------------------------------*/

    Init_decoder(&st);
    Init_read_serial(&rx_state);
    Init_WMOPS_counter();

    /*-------------------------------------------------------------*
     *           Loop for each "L_FRAME" speech data               *
     *-------------------------------------------------------------*/

    fprintf(stderr, "\n --- Running ---\n");

    frame = 0;
    while ((nb_bits = Read_serial(f_serial, prms, &frame_type, &mode, rx_state, bitstreamformat))
!= 0)
    {
       Reset_WMOPS_counter();
```

Lines 175-176:

```
    Close_decoder(st);
    Close_read_serial(rx_state);
    fclose(f_serial);
```

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

Tdoc ⌘S4-030143

CR-Form-v7

CHANGE REQUEST

⌘ **26.173** **CR** **016** ⌘**rev** **-** ⌘ Current version: **5.5.0** ⌘

*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:	⌘	Correction for handling of RX_NO_DATA frames
Source:	⌘	TSG SA WG4
Work item code:	⌘	AMRWB
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:	⌘	Frames of type RX_NO_DATA are not handled appropriately in the AMR-WB speech decoder. The flag “unusable_frame” is set to false for this frametype although RX_NO_DATA frames do not contain any useful information.
Summary of change:	⌘	For frames of type RX_NO_DATA, the flag “unusable_frame” is set to true now.
Consequences if not approved:	⌘	Possible speech degradations if NO_DATA frames are received without preceding SID_FIRST frame.

Clauses affected:	⌘	Source file: dec_main.c

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

Other comments:	⌘	

1. How the code is changed in the file *dec_main.c*

Lines 242-260:

```
    /* SPEECH action state machine  */
    test();test();
    if ((sub(frame_type, RX_SPEECH_BAD) == 0) ||
        (sub(frame_type, RX_NO_DATA) == 0) ||
        (sub(frame_type, RX_SPEECH_PROBABLY_DEGRADED) == 0))
    {
        /* bfi for all index, bits are not usable */
        bfi = 1;                            move16();
        unusable_frame = 0;                 move16();
    } else if ((sub(frame_type, RX_NO_DATA) == 0) ||
               (sub(frame_type, RX_SPEECH_LOST) == 0))
    {
        /* bfi only for lsf, gains and pitch period */
        bfi = 1;                            move16();
        unusable_frame = 1;                 move16();
    } else
    {
        bfi = 0;                            move16();
        unusable_frame = 0;                 move16();
    }
```