Technical Specification Group Services and System Aspects
Meeting #14, Kyoto, Japan, 17-20 December 2001

***TSGS#14(01)0699***

**Source:** **TSG-SA WG4**

**Title: CRs to TS 26.173 on "Incorrect mode usage during DTX" and "Correction of homing function for 23.85 kbit/s mode" (Release 5)**

**Document for:** **Approval**

**Agenda Item:** **7.4.3**

The following CRs, agreed at the TSG-SA WG4 meeting #19, are presented to TSG SA #14 for approval.

| Spec | CR | Rev | Phase | Subject | Cat | Vers | WG | Meeting | S4 doc |
|---|---|---|---|---|---|---|---|---|---|
| 26.173 | 009 | | REL-5 | Incorrect mode usage during DTX | F | 5.2.0 | S4 | TSG-SA WG4#19 | S4-010590 |
| 26.173 | 010 | | REL-5 | Correction of homing function for 23.85 kbit/s mode | F | 5.2.0 | S4 | TSG-SA WG4#19 | S4-010591 |

**3GPP TSG-SA4 Meeting #19**
**Tokyo, Japan, December 3-7, 2001**

***Tdoc S4-010590***

*CR-Form-v4*

# CHANGE REQUEST

⌘ **TS 26.173 CR 009** ⌘ ev **-** ⌘ Current version: **5.2.0** ⌘

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

***Proposed change affects:*** ⌘ (U)SIM ☐ ME/UE **X** Radio Access Network ☐ Core Network **X**

| | | |
|---|---|---|
| ***Title:*** | ⌘ | Incorrect mode usage during DTX |
| ***Source:*** | ⌘ | TSG SA WG4 |
| ***Work item code:*** | ⌘ | AMR-WB |
| ***Date:*** | ⌘ | 17-Dec-2001 |
| ***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)*

***Reason for change:*** ⌘
- In case of FRAME_TYPEs NO_DATA and SPEECH_LOST speech mode from received frame is used even it should not have been received. During NO_DATA or SPEECH_LOST frames, the previous mode should be used.
- Encoder MODE_INFO field in the transmitted speech frame incorrectly uses mode 10 (MRDTX) when FRAME_TYPE field has value TX_SID_FIRST or TX_SID_UPDATE. The MODE_INFO field should contain the currently active speech mode (0…8).

***Summary of change:*** ⌘
- Decoder does not use speech mode information in SPEECH_LOST and NO_DATA frames. Previous mode is used instead.
- Encoder sends valid (0…8) speech mode information in SID_FIRST and SID_UPDATE frames.

***Consequences if not approved:*** ⌘
- It will be impossible to make bit exact systems with TS 26.173 if mode information is taken from NO_DATA and SPEECH_LOST frames.
- Parameter bitstream file format is not consistent with TS 26.173 documentation and DTX-information is sent in duplicate fields (MODE_INFO and FRAME_TYPE).

***Clauses affected:*** ⌘
TS26.173 C-code
- bits.c, bits.h
- decoder.c
- coder.c

***Other specs affected:*** ⌘
☐ Other core specifications ⌘
☐ Test specifications
☐ O&M Specifications

***Other comments:*** ⌘

**How to create CRs using this form:**

Comprehensive information and tips about how to create CRs can be found at: http://www.3gpp.org/3G_Specs/CRs.htm. Below is a brief summary:

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.

2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.

3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

# Changes to code

Left column shows the original files, righ column the updated files where changes are marked as red text.

## bits.c

```
void Write_serial(FILE * fp, Word16 prms[], Word16 mode, TX_State *st)

{
   Word16 i, frame_type;
   Word16 stream[SIZE_MAX];

   if (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;


   stream[0] = TX_FRAME_TYPE;
   stream[1] = frame_type;
   stream[2] = mode;
   for (i = 0; i < nb_of_bits[mode]; i++)
   {
      stream[3 + i] = prms[i];
   }

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

   return;
}


/*---------------------------------------------------------*
 * Read_serial -> read serial stream into a file       *
 *---------------------------------------------------------*/

Word16 Read_serial(FILE * fp, Word16 prms[], Word16 * frame_type, Word16 * mode)


{
   Word16 n, type_of_frame_type;

   n = (Word16) fread(&type_of_frame_type, sizeof(Word16), 1, fp);
   n = (Word16) (n + fread(frame_type, sizeof(Word16), 1, fp));
   if (n == 2)


   {
      if (type_of_frame_type == TX_FRAME_TYPE)
      {
         switch (*frame_type)
         {
         case TX_SPEECH:
```

```
void Write_serial(FILE * fp, Word16 prms[], Word16 coding_mode, Word16 mode,
TX_State *st)
{
  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;


  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);

  return;
}


/*---------------------------------------------------------*
 * Read_serial -> read serial stream into a file       *
 *---------------------------------------------------------*/

Word16 Read_serial(FILE * fp, Word16 prms[], Word16 * frame_type, Word16 *
mode)
{
  Word16 n, type_of_frame_type, coding_mode;

  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 (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);
      }
   }
   n = (Word16) (n + fread(mode, sizeof(Word16), 1, fp));

   if (n == 3)
   {
      n = (Word16) fread(prms, sizeof(Word16), nb_of_bits[*mode], fp);
      if (n != nb_of_bits[*mode])


         n = 0;
   }
   return (n);
}
```

```
            *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);
}
```

# bits.h

```
void Write_serial(FILE * fp, Word16 prms[], Word16 mode, TX_State *st);
```

```
void Write_serial(FILE * fp, Word16 prms[], Word16 coding_mode, Word16 mode,
TX_State *st);
```

# decoder.c

```
int main(int argc, char *argv[])
{
   FILE *f_serial;                  /* File of serial bits for transmission  */
   FILE *f_synth;                   /* File of speech data                   */

   Word16 synth[L_FRAME16k];        /* Buffer for speech @ 16kHz             */
   Word16 prms[NB_BITS_MAX];

   Word16 nb_bits, mode, frame_type, frame_length;
   Word16 reset_flag = 0;
   Word16 reset_flag_old = 1;

   Word16 i;
   long frame;

   void *st;

    fprintf(stderr, "\n");
    fprintf(stderr, " ================================================
==========================\n");



    fprintf(stderr, " 3GPP AMR Wideband Codec, September 7, 2001. Version %s.\n"
, CODEC_VERSION);


    fprintf(stderr, " ================================================
==========================\n");



    fprintf(stderr, "\n");


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

   if (argc != 3)
   {
      fprintf(stderr, "Usage : decoder  bitstream_file  synth_file\n");
      fprintf(stderr, "\n");
      fprintf(stderr, "Format for bitstream_file:\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.\n");
```

```
int main(int argc, char *argv[])
{
   FILE *f_serial;                  /* File of serial bits for transmission  */
   FILE *f_synth;                   /* File of speech data                   */

   Word16 synth[L_FRAME16k];        /* Buffer for speech @ 16kHz             */
   Word16 prms[NB_BITS_MAX];

   Word16 nb_bits, mode, frame_type, frame_length;
   Word16 reset_flag = 0;
   Word16 reset_flag_old = 1;
   Word16 mode_old = 0;
   Word16 i;
   long frame;

   void *st;

    fprintf(stderr, "\n");


    fprintf(stderr, "
============================================================
============\n");

    fprintf(stderr, " 3GPP AMR Wideband Codec, September 7, 2001. Version
%s.\n", CODEC_VERSION);

    fprintf(stderr, "
============================================================
============\n");
    fprintf(stderr, "\n");


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

   if (argc != 3)
   {
      fprintf(stderr, "Usage : decoder  bitstream_file  synth_file\n");
      fprintf(stderr, "\n");
      fprintf(stderr, "Format for bitstream_file:\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.\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);
}
/* 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_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)) != 0)
{
    Reset_WMOPS_counter();

    frame++;

    fprintf(stderr, " Frames processed: %ld\r", frame);




    /* if homed: check if this frame is another homing frame */
    if (reset_flag_old == 1)
    {
        /* only check until end of first subframe */
        reset_flag = decoder_homing_frame_test_first(prms, mode);
    }


    /* produce encoder homing frame if homed & input=decoder homing frame */
    if ((reset_flag != 0) && (reset_flag_old != 0))
    {
        for (i = 0; i < L_FRAME16k; i++)
        {
            synth[i] = EHF_MASK;
        }
    } else
    {
        decoder(mode, prms, synth, &frame_length, st, frame_type);
    }

    for (i = 0; i < L_FRAME16k; i++)   /* Delete the 2 LSBs (14-bit output) */
    {
        synth[i] = (Word16) (synth[i] & 0xfffC);      logic16(); move16();
    }

    fwrite(synth, sizeof(Word16), L_FRAME16k, f_synth);

    WMOPS_output((Word16) (mode == MRDTX));

    /* if not homed: check whether current frame is a homing frame */
    if (reset_flag_old == 0)
    {
        /* check whole frame */
        reset_flag = decoder_homing_frame_test(prms, mode);
    }
    /* reset decoder if current frame is a homing frame */
    if (reset_flag != 0)
    {
        Reset_decoder(st, 1);
```

```
    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);
}
/* 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_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)) != 0)
{
    Reset_WMOPS_counter();

    frame++;

    fprintf(stderr, " Frames processed: %ld\r", frame);

    if ((frame_type == RX_NO_DATA) | (frame_type == RX_SPEECH_LOST))
    {
      mode = mode_old;
      reset_flag = 0;
    }
    else
    {
      mode_old = mode;

    /* if homed: check if this frame is another homing frame */
    if (reset_flag_old == 1)
    {
        /* only check until end of first subframe */
        reset_flag = decoder_homing_frame_test_first(prms, mode);
    }
    }

    /* produce encoder homing frame if homed & input=decoder homing frame */
    if ((reset_flag != 0) && (reset_flag_old != 0))
    {
        for (i = 0; i < L_FRAME16k; i++)
        {
            synth[i] = EHF_MASK;
        }
    } else
    {
        decoder(mode, prms, synth, &frame_length, st, frame_type);
    }

    for (i = 0; i < L_FRAME16k; i++)   /* Delete the 2 LSBs (14-bit output) */
    {
        synth[i] = (Word16) (synth[i] & 0xfffC);      logic16(); move16();
    }

    fwrite(synth, sizeof(Word16), L_FRAME16k, f_synth);

    WMOPS_output((Word16) (mode == MRDTX));

    /* if not homed: check whether current frame is a homing frame */
    if (reset_flag_old == 0)
    {
        /* check whole frame */
        reset_flag = decoder_homing_frame_test(prms, mode);
    }
    /* reset decoder if current frame is a homing frame */
    if (reset_flag != 0)
    {
        Reset_decoder(st, 1);
```

```
      }
      reset_flag_old = reset_flag;

   }

   Close_decoder(st);

   exit(0);
}
```

```
      }
      reset_flag_old = reset_flag;

   }

   Close_decoder(st);

   exit(0);
}
```

# coder.c

```
int main(int argc, char *argv[])
{
   FILE *f_speech;                /* File of speech data                 */
   FILE *f_serial;                /* File of serial bits for transmission  */
   FILE *f_mode = NULL;           /* File of modes for each frame     */

   Word16 signal[L_FRAME16k];     /* Buffer for speech @ 16kHz           */
   Word16 prms[NB_BITS_MAX];

   Word16 coding_mode, nb_bits, allow_dtx, mode_file, mode = 0, i;
   Word16 reset_flag;
   long frame;
   char Mode[2] = "0";

   void *st;
   TX_State *tx_state;

   fprintf(stderr, "\n");
   fprintf(stderr, " ================================================
==========================\n");



   fprintf(stderr, " 3GPP AMR Wideband Codec, September 7, 2001. 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) 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:\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.\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)
```

```
int main(int argc, char *argv[])
{
   FILE *f_speech;                /* File of speech data                 */
   FILE *f_serial;                /* File of serial bits for transmission  */
   FILE *f_mode = NULL;           /* File of modes for each frame     */

   Word16 signal[L_FRAME16k];     /* Buffer for speech @ 16kHz           */
   Word16 prms[NB_BITS_MAX];

   Word16 coding_mode, nb_bits, allow_dtx, mode_file, mode = 0, i;
   Word16 reset_flag;
   long frame;
   char Mode[2] = "0";

   void *st;
   TX_State *tx_state;

   fprintf(stderr, "\n");

   fprintf(stderr, "
============================================================
=============\n");

   fprintf(stderr, " 3GPP AMR Wideband Codec, September 7, 2001. 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) 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:\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.\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++;
   }
   mode_file = 0;
   if (strcmp(argv[1], "-modefile") == 0)
   {
      mode_file = 1;
      argv++;
      if ((f_mode = fopen(argv[1], "r")) == NULL)
      {
         fprintf(stderr, "Error opening input file  %s !!\n", argv[1]);
         exit(0);
      }
      fprintf(stderr, "Mode file:  %s\n", argv[1]);
   } else
   {
      mode = (Word16) atoi(argv[1]);
      if ((mode < 0) || (mode > 8))
      {
         fprintf(stderr, " error in bit rate mode %d: use 0 to 8\n", mode);
         exit(0);
      } else
         nb_bits = nb_of_bits[mode];
   }

   if ((f_speech = fopen(argv[2], "rb")) == NULL)
   {
      fprintf(stderr, "Error opening input file  %s !!\n", argv[2]);
      exit(0);
   }
   fprintf(stderr, "Input speech file:  %s\n", argv[2]);

   if ((f_serial = fopen(argv[3], "wb")) == NULL)
   {
      fprintf(stderr, "Error opening output bitstream file %s !!\n", argv[3]);
      exit(0);
   }
   fprintf(stderr, "Output bitstream file:  %s\n", argv[3]);

   /*--------------------------------------------------------------------------*
    * Initialisation                                                           *
    *--------------------------------------------------------------------------*/

   Init_coder(&st);                    /* Initialize the coder */
   Init_write_serial(&tx_state);
   Init_WMOPS_counter();               /* for complexity calculation */

   /*--------------------------------------------------------------------------*
    * Loop for every analysis/transmission frame.                              *
    *   -New L_FRAME data are read. (L_FRAME = number of speech data per
frame) *


    *   -Conversion of the speech data from 16 bit integer to real             *
    *   -Call coder to encode the speech.                                      *
    *   -The compressed serial output stream is written to a file.             *
    *--------------------------------------------------------------------------*/

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

   frame = 0;

   while (fread(signal, sizeof(Word16), L_FRAME16k, f_speech) == L_FRAME16k)
   {
      Reset_WMOPS_counter();

      if (mode_file)
      {
         if (fread(Mode, sizeof(char), 1, f_mode) != 1)
         {
            fprintf(stderr, "\nend of mode control file reached\n");
            fprintf(stderr, "From now on using mode: %d kbit/s.\n", nb_of_bits[mode]);
         }
         mode = (Word16) atoi(Mode);
      }
      coding_mode = mode;

      frame++;
      fprintf(stderr, " Frames processed: %ld\r", frame);

      /* check for homing frame */
      reset_flag = encoder_homing_frame_test(signal);

      for (i = 0; i < L_FRAME16k; i++)   /* Delete the 2 LSBs (14-bit input) */
      {
         signal[i] = (Word16) (signal[i] & 0xfffC);      logic16(); move16();
      }

      coder(&coding_mode, signal, prms, &nb_bits, st, allow_dtx);
```

```
   {
      allow_dtx = 1;
      argv++;
   }
   mode_file = 0;
   if (strcmp(argv[1], "-modefile") == 0)
   {
      mode_file = 1;
      argv++;
      if ((f_mode = fopen(argv[1], "r")) == NULL)
      {
         fprintf(stderr, "Error opening input file  %s !!\n", argv[1]);
         exit(0);
      }
      fprintf(stderr, "Mode file:  %s\n", argv[1]);
   } else
   {
      mode = (Word16) atoi(argv[1]);
      if ((mode < 0) || (mode > 8))
      {
         fprintf(stderr, " error in bit rate mode %d: use 0 to 8\n", mode);
         exit(0);
      } else
         nb_bits = nb_of_bits[mode];
   }

   if ((f_speech = fopen(argv[2], "rb")) == NULL)
   {
      fprintf(stderr, "Error opening input file  %s !!\n", argv[2]);
      exit(0);
   }
   fprintf(stderr, "Input speech file:  %s\n", argv[2]);

   if ((f_serial = fopen(argv[3], "wb")) == NULL)
   {
      fprintf(stderr, "Error opening output bitstream file %s !!\n", argv[3]);
      exit(0);
   }
   fprintf(stderr, "Output bitstream file:  %s\n", argv[3]);

   /*--------------------------------------------------------------------------*
    * Initialisation                                                           *
    *--------------------------------------------------------------------------*/

   Init_coder(&st);                    /* Initialize the coder */
   Init_write_serial(&tx_state);
   Init_WMOPS_counter();               /* for complexity calculation */

   /*--------------------------------------------------------------------------*
    * Loop for every analysis/transmission frame.                              *


    *   -New L_FRAME data are read. (L_FRAME = number of speech data per
frame) *
    *   -Conversion of the speech data from 16 bit integer to real             *
    *   -Call coder to encode the speech.                                      *
    *   -The compressed serial output stream is written to a file.             *
    *--------------------------------------------------------------------------*/

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

   frame = 0;

   while (fread(signal, sizeof(Word16), L_FRAME16k, f_speech) == L_FRAME16k)
   {
      Reset_WMOPS_counter();

      if (mode_file)
      {
         if (fread(Mode, sizeof(char), 1, f_mode) != 1)
         {
            fprintf(stderr, "\nend of mode control file reached\n");
            fprintf(stderr, "From now on using mode: %d kbit/s.\n", nb_of_bits[mode]);
         }
         mode = (Word16) atoi(Mode);
      }
      coding_mode = mode;

      frame++;
      fprintf(stderr, " Frames processed: %ld\r", frame);

      /* check for homing frame */
      reset_flag = encoder_homing_frame_test(signal);

      for (i = 0; i < L_FRAME16k; i++)   /* Delete the 2 LSBs (14-bit input) */
      {
         signal[i] = (Word16) (signal[i] & 0xfffC);      logic16(); move16();
      }

      coder(&coding_mode, signal, prms, &nb_bits, st, allow_dtx);
```

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

        WMOPS_output((Word16) (coding_mode == MRDTX));

        /* perform homing if homing frame was detected at encoder input */
        if (reset_flag != 0)
        {
           Reset_encoder(st, 1);
        }
     }

     Close_coder(st);

     exit(0);
}
```

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

        WMOPS_output((Word16) (coding_mode == MRDTX));

        /* perform homing if homing frame was detected at encoder input */
        if (reset_flag != 0)
        {
           Reset_encoder(st, 1);
        }
     }

     Close_coder(st);

     exit(0);
}
```

**3GPP TSG-SA4 Meeting #19**
**Tokyo, Japan, December 3-7, 2001**

***Tdoc S4-010591***

*CR-Form-v4*

# CHANGE REQUEST

⌘ **TS 26.173 CR 010** ⌘ ev **-** ⌘ Current version: **5.2.0** ⌘

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

***Proposed change affects:*** ⌘ (U)SIM [ ] ME/UE [X] Radio Access Network [ ] Core Network [X]

| | | |
|---|---|---|
| ***Title:*** | ⌘ | Correction of decoder homing function for 23.85 kbit/s mode |
| ***Source:*** | ⌘ | TSG SA WG4 |
| ***Work item code:*** | ⌘ | AMR-WB |
| ***Date:*** | ⌘ | 17-Dec-2001 |
| ***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)*

| | | |
|---|---|---|
| ***Reason for change:*** | ⌘ | • In case of 23.85 kbit/s speech mode operating with DTX option, homing function is not working. The initial state after reset is different for the mode 23.85 kbit/s with and without DTX. This is caused by high band energy interpolation during the hangover period. |
| ***Summary of change:*** | ⌘ | • When operating with 23.85 kbit/s speech mode in the decoder, the high band energy bits (16bit/frame) are ignored when homing frame is checked. |
| ***Consequences if not approved:*** | ⌘ | • Homing function is not working in case of 23.85 kbit/s speech mode operating with DTX option. |

| | | |
|---|---|---|
| ***Clauses affected:*** | ⌘ | TS 26.173 C-code<br>- homing.c<br>- homing.tab<br>3GPP TS 26.173 table 9 |
| ***Other specs affected:*** | ⌘ | [ ] Other core specifications ⌘<br>[ ] Test specifications<br>[ ] O&M Specifications |
| ***Other comments:*** | ⌘ | Note, that the whole table 9 has been changed (homing values for all the modes) in this CR, despite the correction of this CR applies only to the 23.85 kbit/s mode. This is because S4 already produced CR 26.173-005 rev 1, in Tdoc SP-010307, which changed the values for all the modes; this CR was approved at TSG SA 12. Although the changes were implemented in the C-code, and do exist in the latest version 5.2.0, unfortunately the changes were not implemented in the "textual" part of the specification; this is why the changes are re-done here, to correct all the homing values in table 9 of TS 26.173 version 5.3.0. |

**How to create CRs using this form:**

Comprehensive information and tips about how to create CRs can be found at: http://www.3gpp.org/3G_Specs/CRs.htm. Below is a brief summary:

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.

2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.

3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

# How 3GPP TS 26.173 V5.2.0 is changed

## Before the change

**Table 9: Table values for the decoder homing frame in 15-bit-long format for different modes**

| Mode | Value (MSB=b0) |
|---|---|
| 0 | 25351, 4331, 515, 15620, 20992, 0, 0, 0, 0 |
| 1 | 25351, 14010, 26489, 30912, 5254, 3459, 0, 0, 0, 0, 0, 0 |
| 2 | 25351, 14010, 29177, 18070, 19971, 3968, 32492, 8430, 13280, 0, 0, 0, 0, 0, 0, 0, 0 |
| 3 | 25351, 14010, 1912, 16326, 25140, 16384, 502, 15167, 1772, 11512, 0, 0, 0, 0, 0, 0, 0, 0, 0 |
| 4 | 25351, 14010, 1912, 30593, 14594, 19990, 864, 4635, 20446, 27456, 21310, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 |
| 5 | 25351, 14010, 19995, 14446, 6159, 7329, 20752, 4228, 19488, 24383, 364, 20124, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 |
| 6 | 25351, 14010, 3567, 560, 32536, 20534, 5139, 16384, 26161, 18755, 20444, 22173,12623, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 |
| 7 | 25351, 14010, 2912, 28827, 15347, 28610, 9853, 1316, 30720, 786, 32259, 13279,14336, 29152, 23302, 20352, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0 |
| 8 | 025351, 14010, 1601, 16734, 7923, 15017, 5450, 5477, 5760, 2187, 1534, 12142, 30894, 13419, 13141, 2376, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 |

## After the change (this CR + CR 005)

**Table 9: Table values for the decoder homing frame in 15-bit-long format for different modes**

| Mode | Value (MSB=b0) |
|---|---|
| 0 | 3168, 29954, 29213, 16121, 64, 13440, 30624, 16430, 19008 |
| 1 | 3168, 31665, 9943, 9123, 15599, 4358, 20248, 2048, 17040, 27787, 16816, 13888 |
| 2 | 3168, 31665, 9943, 9128, 3647, 8129, 30930, 27926, 18880, 12319, 496, 1042, 4061, 20446, 25629, 28069, 13948 |
| 3 | 3168, 31665, 9943, 9131, 24815, 655, 26616, 26764, 7238, 19136, 6144, 88, 4158, 25733, 30567, 30494, 221, 20321, 17823 |
| 4 | 3168, 31665, 9943, 9131, 24815, 700, 3824, 7271, 26400, 9528, 6594, 26112, 108, 2068, 12867, 16317, 23035, 24632, 7528, 1752, 6759, 24576 |
| 5 | 3168, 31665, 9943, 9135, 14787, 14423, 30477, 24927, 25345, 30154, 916, 5728, 18978, 2048, 528, 16449, 2436, 3581, 23527, 29479, 8237, 16810, 27091, 19052, 0 |
| 6 | 3168, 31665, 9943, 9129, 8637, 31807, 24646, 736, 28643, 2977, 2566, 25564, 12930, 13960, 2048, 834, 3270, 4100, 26920, 16237, 31227, 17667, 15059, 20589, 30249, 29123, 0 |
| 7 | 3168, 31665, 9943, 9132, 16748, 3202, 28179, 16317, 30590, 15857, 19960, 8818, 21711, 21538, 4260, 16690, 20224, 3666, 4194, 9497, 16320, 15388, 5755, 31551, 14080, 3574, 15932, 50, 23392, 26053, 31216 |
| 8 | 3168, 31665, 9943, 9134, 24776, 5857, 18475, 28535, 29662, 14321, 16725, 4396, 29353, 10003, 17068, 20504, 720, 0, 8465, 12581, 28863, 24774, 9709, 26043, 7941, 27649, 13965, 15236, 18026, 22047, 16681, 3968 |

# Changes to code

Left column shows the original files, righ column the updated files where changes are marked as red text.

## homing.tab

```
static const Word16 dfh_M24k[PRMN_24k] =

{

 3168, 31665,  9943,  9134,

 24776,  5857, 18475, 28535,

 29662, 14321, 18261,  4396,

 29353, 10003, 17068, 20504,

  720,     0,  8465, 12581,

 28863, 24774,  9709, 26043,

  7957, 27649, 13965, 15236,

 18026, 22047, 16681,  3968

};
```

```
static const Word16 dfh_M24k[PRMN_24k] =

{

 3168, 31665,  9943,  9134,

 24776,  5857, 18475, 28535,

 29662, 14321, 16725,  4396,

 29353, 10003, 17068, 20504,

  720,     0,  8465, 12581,

 28863, 24774,  9709, 26043,

  7941, 27649, 13965, 15236,

 18026, 22047, 16681,  3968

};
```

## homing.c

```
Word16 encoder_homing_frame_test(Word16 input_frame[])
{
   Word16 i, j = 0;

   /* check 320 input samples for matching EHF_MASK: defined in e_homing.h */
   for (i = 0; i < L_FRAME16k; i++)
   {
      j = (Word16) (input_frame[i] ^ EHF_MASK);

      if (j)
         break;
   }

   return (Word16) (!j);
}

static Word16 dhf_test(Word16 input_frame[], Word16 mode, Word16 nparms)
{
   Word16 i, j, tmp, shift;
   Word16 param[DHF_PARMS_MAX];
   Word16 *prms;

   prms = input_frame;
   j = 0;
   i = 0;

   if (sub(mode, MRDTX) != 0)
   {


      /* convert the received serial bits */
      tmp = sub(nparms, 15);
      while (sub(tmp, j) > 0)
      {
         param[i] = Serial_parm(15, &prms);
         j = add(j, 15);
         i = add(i, 1);
      }
      tmp = sub(nparms, j);
      param[i] = Serial_parm(tmp, &prms);
      shift = sub(15, tmp);
      param[i] = shl(param[i], shift);
```

```
Word16 encoder_homing_frame_test(Word16 input_frame[])
{
   Word16 i, j = 0;

   /* check 320 input samples for matching EHF_MASK: defined in e_homing.h */
   for (i = 0; i < L_FRAME16k; i++)
   {
      j = (Word16) (input_frame[i] ^ EHF_MASK);

      if (j)
         break;
   }

   return (Word16) (!j);
}

static Word16 dhf_test(Word16 input_frame[], Word16 mode, Word16 nparms)
{
   Word16 i, j, tmp, shift;
   Word16 param[DHF_PARMS_MAX];
   Word16 *prms;

   prms = input_frame;
   j = 0;
   i = 0;

   if (sub(mode, MRDTX) != 0)
   {
  if (sub(mode, MODE_24k) != 0)
  {
   /* convert the received serial bits */
   tmp = sub(nparms, 15);
   while (sub(tmp, j) > 0)
   {
    param[i] = Serial_parm(15, &prms);
     j = add(j, 15);
    i = add(i, 1);
   }
   tmp = sub(nparms, j);
   param[i] = Serial_parm(tmp, &prms);
   shift = sub(15, tmp);
   param[i] = shl(param[i], shift);
   }
```

```
else
{
 /*If mode is 23.85Kbit/s, remove high band energy bits */
 for (i = 0; i < 10; i++)
 {
  param[i] = Serial_parm(15, &prms);
 }
  param[10] = Serial_parm(15, &prms) & 0x61FF;
 for (i = 11; i < 17; i++)
 {
  param[i] = Serial_parm(15, &prms);
 }
  param[17] = Serial_parm(15, &prms) & 0xE0FF;
 for (i = 18; i < 24; i++)
 {
  param[i] = Serial_parm(15, &prms);
 }
  param[24] = Serial_parm(15, &prms) & 0x7F0F;
 for (i = 25; i < 31; i++)
 {
  param[i] = Serial_parm(15, &prms);
 }
  tmp = Serial_parm(8, &prms);
  param[31] = shl(tmp,7);
  shift=0;
}
```

```
    /* check if the parameters matches the parameters of the corresponding decoder homing frame */
```

```
    /* check if the parameters matches the parameters of the corresponding decoder
homing frame */
    tmp = i;
    j = 0;
    for (i = 0; i < tmp; i++)
    {
       j = (Word16) (param[i] ^ dhf[mode][i]);
       if (j)
          break;
    }
    tmp = 0x7fff;
    tmp = shr(tmp, shift);
    tmp = shl(tmp, shift);
    tmp = (Word16) (dhf[mode][i] & tmp);
    tmp = (Word16) (param[i] ^ tmp);
    j = (Word16) (j | tmp);

  }
  else
  {
    j = 1;
  }

  return (Word16) (!j);
}
```

```
    /* check if the parameters matches the parameters of the corresponding decoder

    tmp = i;
    j = 0;
    for (i = 0; i < tmp; i++)
    {
       j = (Word16) (param[i] ^ dhf[mode][i]);
       if (j)
          break;
    }
    tmp = 0x7fff;
    tmp = shr(tmp, shift);
    tmp = shl(tmp, shift);
    tmp = (Word16) (dhf[mode][i] & tmp);
    tmp = (Word16) (param[i] ^ tmp);
    j = (Word16) (j | tmp);

  }
  else
  {
    j = 1;
  }

  return (Word16) (!j);
}
```