ETSI TS 126 230 V11.0.0 (2012-10)

Technical Specification

**Digital cellular telecommunications system (Phase 2+);
Universal Mobile Telecommunications System (UMTS);
Cellular text telephone modem;
Transmitter bit exact C-code
(3GPP TS 26.230 version 11.0.0 Release 11)**

Reference

RTS/TSGS-0426230vb00

Keywords

GSM,UMTS

ETSI

650 Route des Lucioles

F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00   Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C

Association à but non lucratif enregistrée à la

Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:

http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at

http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:

http://portal.etsi.org/chaircor/ETSI_support.asp

Copyright Notification

ETSI

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under http://webapp.etsi.org/key/queryform.asp.

Contents

Foreword

This Technical Specification has been produced by T1P1.

The contents of the present document are subject to continuing work within the 3GPP TSG and may change following formal 3GPP approval. Should the 3GPP TSG modify the contents of this TS, it will be re-released by the 3GPP TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x   the first digit:

1   presented to 3GPP for information;

2   presented to 3GPP for approval;

3   Indicates 3GPP approved document under change control.

y   the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z   the third digit is incremented when editorial only changes have been incorporated in the specification;

0   Scope

This Technical Standard (TS) contains an electronic copy of the ANSI-C code for the Cellular Text Telephone Modem (CTM) for reliable transmission of text telephone text via the speech channel of cellular networks. While CTM is generally usable with text in UCS coding, the example application linked to CTM in this document is limited to use the signals and character set of the Baudot type.

1   Normative references

This TS incorporates by dated and undated reference, provisions from other publications. These normative references are cited at the appropriate places in the text and the publications are listed hereafter. For dated references, subsequent amendments to or revisions of any of these publications apply to this TS only when incorporated in it by amendment or revision. For undated references, the latest edition of the publication referred to applies.

[1]   3GPP TS 26.226: "Cellular text telephone modem; General description".

[2]   ISO/IEC 10646-1: "Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane".

2   Definitions and Abbreviations

For the purposes of this TS, the following abbreviations apply:

CTM	Cellular Text Telephone Modem
FEC	Forward Error Correction
FSK	Frequency Shift Key
HCO	Hearing Carry Over, (individual may be able to hear, but cannot speak) Alternating transmission of speech and text.
PCM	Pulse Code Modulation
RX	Receive
TX	Transmit
TTY	Text Telephone
UCS	Universal Multiple-Octet Coded Character Set
UTF	UCS transformation format
VAD	Voice Activity Detection
VCO	Voice Carry Over, Alternating transmission of speech and text

3   C code structure

This clause gives an overview of the structure of the bit-exact C code and provides an overview of the contents and organization of the C code attached to this document.

The C code has been verified on the following system.

- Sun Microsystems workstations with SUN Solaris^{TM} operating system and the the Gnu C Compiler (gcc version 2.7.2.3) and GNU Make 3.77;

The C code has also been successfully compiled and used in the following environment, with the exception that it cannot be guaranteed that the upper part of the UCS code table in file `ucs_functions.c` will be compiled correctly since it depends on the codepage setting of the environment.

- IBM PC/AT compatible computers with Windows^{TM} NT 4.0 operating system and Microsoft Visual C++ 6.0^{TM} compiler.

3.1   Contents of the C source code

The distributed files with suffix "c" contain the source code and the files with suffix "h" are the header files. All these files are in the root level of the ZIP-archive.

Makefiles are provided for the platforms in which the C code has been verified (listed above). They are called 'Makefile' for GNU Make and 'Makefile.vc' for Microsoft Visual C++^{TM}.

For the Sun Microsystems platform, an example shell script for a transmission via two signal adaptation modules is given in "test_negotiation". For the Microsoft Windows^{TM} platform, no shell script or batch program is provided.

The software can be compiled using the commands

```
make all     or     gmake all      in case of Gnu Make
```

```
nmake /f Makefile.vc                in case of Microsoft Visual C++.
```

The executables are compiled into the directory `./solaris` (in case of Gnu Make) or into the actual directory in case of Microsoft Visual C++^{TM}.

The directory `./patterns` provides the file `baudot.pcm` that serves as input signal for the test script `test_negotiation`. All output data of `test_negotiation` will be stored into the directory `./output`. If required, this directory will be created by `test_negotiation` automatically.

3.2   Program execution

The CTM signal adaptation module is implemented in the execuable `adaptation_switch` (in case of Sun Solaris^{TM} platform) or `adaptation_switch.exe` (in case of the Micorsoft Windows^{TM} platform).

The program should be called like:

```
adaptation_switch -ctmin <file>    -ctmout <file>
                  -baudotin <file> -baudotout <file>
```

using the following parameters:

```
  -ctmin       <input_file>   input file with CTM signal

  -ctmout      <output_file>  output file for CTM signal

  -baudotin    <input_file>   input file with Baudot Tones

  -baudotout   <output_file>  output file for Baudot Tones
  -textout     <text_file>    output text file from CTM receiver (optional)
  -numsamples  <number>       number of samples to process (optional)
  -nonegotiation              disables the negotiation (optional)
```

All files contain 16-bit linear encoded PCM audio samples, which are swapped according to the platform"s endian type (Sun Microsystems platforms use big endian, Intel platforms use little endian). An example file `baudot.pcm` containing a Baudot Code modem signal (big endian) is provided in the subdirectory `./patterns`.

Due to the fact that the signal adaptation module expects a successful negotiation before Baudot Code signals can be converted to CTM signals, the signal adaptation module has to be executed several times in two instances in order to execute a successful negotiation. For the Sun Microsystems platform, a shell script `test_negotiation` is provided for executing the following structure:

```
                        ----------    ctm_forward    ----------
 baudot.pcm --->|           | --------------->|           |---> baudot_out.pcm

                | adapt#1 |                   | adapt#2 |

  /dev/null <---|           | <---------------|           |<--- /dev/zero

                        ----------    ctm_backward   ----------
```
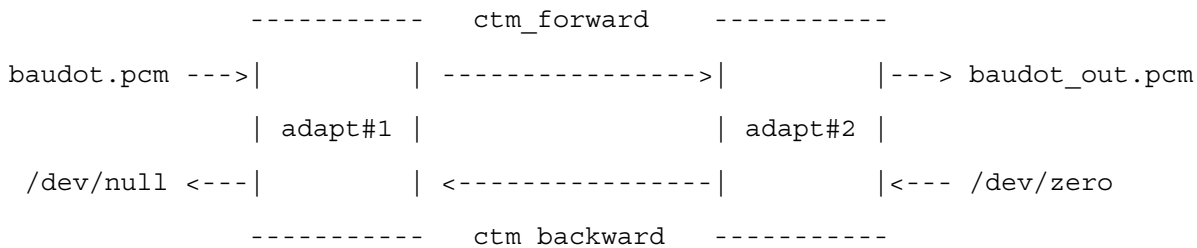
First, the adaptation module #1 is executed. At this first run, the signal ctm_backward is not known. Therefore, the negotiation does not get a positive acknowledge, so that the transmission falls back to Baudot Tones.

Then signal adaptation module #2 is executed for the first time.

After that, adaptation module #1 is executed for the second time. With this second run, the signal ctm_backward is valid. Therefore, the negotiation receives a valid acknowledge, so that CTM signals are transmitted.

At last, adaptation module #2 is executed for the second time. With this run, adaptation module #2 receives a valid CTM signal so that the baudot_out.pcm signal can be generated.

After executing each of the modules twice, the signal baudot_out.pcm is analyzed. This analysis is also performed by the program adaptation_switch. First, the Baudot detector of adaptation_switch is used for this analysis in order to examine whether the regenerated Baudot signal can be decoded correctly. In a second step it is examined whether the regenerated signal still contains any CTM preambles. This investigation is performed by means of the CTM detector that is integrated in adaptation_switch. This last test fails if the CTM detector is able to detect any CTM preamble in the regenerated signal.

During the execution of the script test_negotiation the following text output shall be generated:

```
======================================================
Execute adaptation module #1 (first pass)
======================================================


*********************************************************************

  Cellular Text Telephone Modem (CTM) - Example Implementation for

  Conversion between CTM and Baudot Code (use option -h for help)
*********************************************************************


number of samples to process: 100000


>>> Enquiry Burst generated! <<<

THE>>> Enquiry Burst generated! <<<

 >>> Enquiry Burst generated! <<<

CELL


======================================================
```

```
Execute adaptation module #2 (first pass)

========================================================


********************************************************************

   Cellular Text Telephone Modem (CTM) - Example Implementation for

   Conversion between CTM and Baudot Code (use option -h for help)

********************************************************************


>>> CTM from far-end detected! <<<

>>> Enquiry From Far End Detected! <<<

THE>>> Enquiry From Far End Detected! <<<

 >>> Enquiry From Far End Detected! <<<

CELL


======================================================

Execute adaptation module #1 (second pass)

======================================================


********************************************************************

   Cellular Text Telephone Modem (CTM) - Example Implementation for

   Conversion between CTM and Baudot Code (use option -h for help)

********************************************************************


>>> Enquiry Burst generated! <<<

THE>>> CTM from far-end detected! <<<

 CELLULAR TEXT TELEPHONE MODEM (CTM) ALLOWS RELIABLE

TRANSMISSION OF A TEXT TELEPHONE CONVERSATION ALTERNATING

WITH A SPEECH CONVERSATION THROUGH THE EXISTING SPEECH

COMMUNICATION PATHS IN CELLULAR MOBILE PHONE SYSTEMS.

THIS RELIABILITY IS ACHIEVED BY AN IMPROVED MODULATION

TECHNIQUE, INCLUDING ERROR PROTECTION, INTERLEAVING AND

SYNCHRONIZATION.


======================================================

Execute adaptation module #2 (second pass)
```

ETSI

==

```
*********************************************************************

  Cellular Text Telephone Modem (CTM) - Example Implementation for

  Conversion between CTM and Baudot Code (use option -h for help)

*********************************************************************


>>> CTM from far-end detected! <<<

>>> Enquiry From Far End Detected! <<<

THE CELLULAR TEXT TELEPHONE MODEM (CTM) ALLOWS RELIABLE

TRANSMISSION OF A TEXT TELEPHONE CONVERSATION ALTERNATING

WITH A SPEECH CONVERSATION THROUGH THE EXISTING SPEECH

COMMUNICATION PATHS IN CELLULAR MOBILE PHONE SYSTEMS.

THIS RELIABILITY IS ACHIEVED BY AN IMPROVED MODULATION

TECHNIQUE, INCLUDING ERROR PROTECTION, INTERLEAVING AND

SYNCHRONIZATION.


=====================================================================

Now we try to decode the regenerated Baudot signal. The text message

shall be decoded completely now...

=====================================================================


*********************************************************************

  Cellular Text Telephone Modem (CTM) - Example Implementation for

  Conversion between CTM and Baudot Code (use option -h for help)

*********************************************************************


THE CELLULAR TEXT TELEPHONE MODEM (CTM) ALLOWS RELIABLE

TRANSMISSION OF A TEXT TELEPHONE CONVERSATION ALTERNATING

WITH A SPEECH CONVERSATION THROUGH THE EXISTING SPEECH

COMMUNICATION PATHS IN CELLULAR MOBILE PHONE SYSTEMS.

THIS RELIABILITY IS ACHIEVED BY AN IMPROVED MODULATION

TECHNIQUE, INCLUDING ERROR PROTECTION, INTERLEAVING AND

SYNCHRONIZATION.
```

```
=======================================================================

Testing whether the regenerated Baudot signal is free of CTM headers.

No CTM burst shall be detected now...

=======================================================================



**********************************************************************

  Cellular Text Telephone Modem (CTM) - Example Implementation for

  Conversion between CTM and Baudot Code (use option -h for help)

**********************************************************************
```

3.3 Code hierarchy

This section gives an overview of the hierarchy how the functions are used in the signal adaptation module. All standard C functions: printf(), fwrite(), etc. have been omitted. Also, all functions related to the asynchronous transfer between the signal processing functions by means of FIFO buffers (Shortint_fifo_push, Shortint_fifo_pop, etc.) are not listed in the charts.

The following functions are not part of the actual CTM bit exact specification but are included to allow demonstration of CTM in a Baudot environment:

- init_baudot_tonedemod
- init_baudot_tonemod
- baudot_tonedemod
- convertUCScode2char
- convertChar2TTYcode
- baudot_tonemod
- convertTTYcode2char
- convertChar2UCScode

3.3.1 Initialization routines

The following functions are called for the initialization of the signal adaptation module.

init_baudot_tonedemod		
init_baudot_tonemod		
init_ctm_transmitter	init_interleaver	generate_scambling_sequence
		m_sequence
	init_tonemod	
	conv_encoder_init	
	generate_resync_sequence	m_sequence
	calc_mute_positions	
init_ctm_receiver	init_tonedemod	sin_fip
	viterbi_init	
	calc_mute_positions	
	init_deinterleaver	generate_scambling_sequence
	init_wait_for_sync	m_sequence
		generate_scambling_sequence

3.3.2   Signal Processing Functions

The following functions are called during the main signal processing loop.

baudot_tonedemod	iir_filt	
ctm_receiver	tonedemod	rotate_right
		rotate_left
	wait_for_sync	
	reinit_deinterleaver	
	viterbi_reinit	
	diag_deinterleaver	
	shift_deinterleaver	
	mutingRequired	
	viterbi_exec	
	reinit_wait_for_sync	
	reinit_deinterleaver	
	viterbi_reinit	
	transformUTF2UCS	
convertUCScode2char		
convertChar2TTYcode		
baudot_tonemod		
convertTTYcode2char		
convertChar2UCScode		
ctm_transmitter	transformUCS2UTF	
	reinit_interleaver	
	conv_encoder_exec	
	mutingRequired	
	diag_interleaver	
	diag_interleaver_flush	
	tonemod	

3.4 Description of global constants used in the C-code

The following constants are defined in the file `ctm_defines.h`

Constant	Value	Description
MAX_IDLE_SYMB	5	Number of Idle Symbols at End of Burst
CHC_RATE	4	Rate of the Error Protection
CHC_K	5	Constraint Length of the Error Protection
SYMB_LEN	40	Length of one CTM symbol
LENGTH_TONE_VEC	1	frame size
LENGTH_TX_BITS	8	number of bits per 20 ms frame
BITS_PER_SYMB	8	bits per symbol
NCYCLES_0	2	Number of periods for symbol #0
NCYCLES_1	3	Number of periods for symbol #1
NCYCLES_2	4	Number of periods for symbol #2
NCYCLES_3	5	Number of periods for symbol #3
THRESHOLD_RELIABILITY_FOR_SUPPRESSING_OUTPUT	100	Characters with lower reliability are suppressed
THRESHOLD_RELIABILITY_FOR_XCORR	200	Bits with lower reliability don"t contribute to xcorr
THRESHOLD_RELIABILITY_FOR_GOING_OFFLINE	100	Threshold for regarding a bit as unreliable
MAX_NUM_UNRELIABLE_GROSS_BITS	400	Receiver goes offline after 400 unreliable bits
NUM_BITS_GUARD_INTERVAL	6	Number of muted bits between two bursts
WAIT_SYNC_REL_THRESHOLD_0	20316	(=0.62) rel. threshold for preamble
WAIT_SYNC_REL_THRESHOLD_1	17039	(=0.52) rel. threshold for preamble
WAIT_SYNC_REL_THRESHOLD_2	23065	(=0.71) dto. in case that RX is already online
RESYNC_REL_THRESHOLD	26542	Threshold for Resynchronization (=0.81)
GUARD_BIT_SYMBOL	10	magic number indicating that a bit shall be muted
intlvB	8	Interleaver block length (number of rows)
intlvD	2	Interleaver block distance (interlace factor)
demodSyncLns	1	Number of demodulator sync lines
deintSyncLns	0	Number of deinterleaver sync lines
IDLE_SYMB	0x16	UCS code for Idle Symbol
ENQU_SYMB	0x05	UCS code for Enquiry Symbol
ENQUIRY_TIMEOUT	3040	number of 20-ms frames for negotiation
NUM_ENQUIRY_BURSTS	3	number of enquiry attempts
NUM_MUTE_ROWS	4	Number of Intl. rows that shall be muted
RESYNC_SEQ_LENGTH	32	length of the resynchronization sequence, must be a multiple of 8
NUM_BITS_BETWEEN_RESYNC	352	Distance between two resync sequences, the value `NUM_BITS_BETWEEN_RESYNC+RESYNC_SEQ_LENGTH` must be a multiple of CHC_RATE, intlvB, and BITS_PER_CHAR, and must be greater than `intlvB*((intlvB-1)*intlvD+NUM_MUTE_ROWS`
BAUDOT_NUM_INFO_BITS	5	number of information bits per Baudot character

`BAUDOT_SHIFT_FIGURES`	27	code of shift to figures symbol
`BAUDOT_SHIFT_LETTERS`	31	code of shift to letters symbol
`BAUDOT_BIT_DURATION`	176	must be 176 (for 45.45 baud) or 160 (50 baud)
`BAUDOT_LP_FILTERORDER`	1	Order of the low-pass filters in function `baudot_tonedemod()`
`BAUDOT_BP_FILTERORDER`	2	Order of the according band-pass filters, must be equal to `2*BAUDOT_BP_FILTERORDER`

3.5   Type Definitions

In order to make the C code platform-independent, the following type definitions have been used, which are defined in `typedefs.h`:

```
defined type   meaning          corresponding constants

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

Char           character        (none)

Bool           boolean          true, false

Shortint       16-bit signed    minShortint,  maxShortint
UShortint      16-bit unsigned  minUShortint, maxUShortint

Longint        32-bit signed    minLongint,   maxLongint
ULongint       32-bit unsigned  minULongint,  maxULongint
```

3.6   Functions of the C Code

```
void baudot_tonedemod(Shortint* toneVec, Shortint numSamples,
                  fifo_state_t* ptrOutFifoState,
                  baudot_tonedemod_state_t* state);

Purpose:         Demodulator for Baudot Tones
Defined in:      baudot_functions.c

Input Variables:
toneVec                      Vector containing the input audio signal
numSamples                   Length of toneVec

Input/Output Variables:
ptrOutFifoState              Pointer to the state of the output shift register
                             containing the demodulated TTY codes
state                        Pointer to the state variable of baudot_tonedemod()
```

```
void baudot_tonemod(Shortint  inputTTYcode,
                Shortint *outputToneVec,
                Shortint  lengthToneVec,
                Shortint *ptrNumBitsStillToModulate,
                baudot_tonemod_state_t* state);

Purpose:         Modulator for Baudot Tones
Defined in:      baudot_functions.c
```

```
Input Variables:
inputTTYcode                TTY code of the character that has to be modulated.
                            inputTTYcode must be in the range 0...63, otherwise
                            it is assumed that there is no character to
                            modulate.
lengthToneVec               Indicates how many samples have to be generated.

Output Variables:
outputToneVec               Vector where the output samples are written to.
ptrNumBitsStillToModulate   Indicates how many bits are still in the fifo
                            buffer.


Input/Output Variables:
state                       Pointer to the state variable of baudot_tonedemod()
```

```
void calc_mute_positions(Shortint *mute_positions,
                         Shortint num_rows_to_mute,
                         Shortint start_position,
                         Shortint B,
                         Shortint D);
```

```
Purpose:        Calculation of the indices of the bits that have to be muted
                within one burst. The indices are returned in the vector
                mute_positions.
Defined in:     init_interleaver.c
```

```
Shortint convertChar2ttyCode(char inChar);
```

```
Purpose:        Conversion from character into TTY code
Defined in:     baudot_functions.c
```

```
Input Variables:
inChar                      character that shall be converted
```

```
Return Value:               baudot code of the input or -1 in case that inChar
                            is not valid (e.g. inChar=='\0')
```

```
UShortint convertChar2UCScode(char inChar);
```

```
Purpose:        Conversion from character into UCS code (Universal Multiple-
                Octet Coded Character Set, Row 00 of the Multilingual plane
                according to ISO/IEC 10646-1).  This routine only handles
                characters in the range 0..255 since that is all that is
                required for demonstration of Baudot support.
Defined in:     ucs_functions.c
```

```
Input Variables:
inChar                      character that shall be converted
```

```
Return Value:               UCS code of the input or 0x0016 <IDLE> in case that
                            inChar is not valid (e.g. inChar=='\0')
```

```
char convertTTYcode2char(Shortint ttyCode);
```

Purpose: Conversion from TTY code into Character
Defined in: baudot_functions.c

Input Variables:
ttyCode   Baudot code (must be within the range 0...63) or -1
                              if there is nothing to convert

Return Value:
character (or '\0' if ttyCode is not valid)

```
char convertUCScode2char(UShortint ucsCode);
```

Purpose: Conversion from UCS code into character (Universal Multiple-
                  Octet Coded Character Set, Row 00 of the Multilingual plane
                  according to ISO/IEC 10646-1). This routine only handles
                  characters in the range 0..255 since that is all that is
                  required for demonstration of Baudot support.

Defined in: ucs_functions.c

Input Variables:
ucsCode   UCS code index, must be within the range 0...255

Return Value: character (or '\0' if ucsCode is not valid)

```
void conv_encoder_exec(conv_encoder_t* ptr_state, Shortint* in,
          Shortint  inbits, Shortint* out);
```

Purpose: Execution of the convolutional encoder for error protection
Defined in: conv_encoder.c

Input Variables:
in   Vector with net bits
inbits   Number of valid net bits in vector in

Output variables:
out   Vector with the encoded gross bits. The gross bits
                  are either 0 or 1. The vector out must have at
                  least CHC_RATE*inbits elements.

Input/output variables:
*ptr_state   state variable of the encoder

```
void conv_encoder_init(conv_encoder_t* ptr_state);
```

Purpose: Initialization of the convolutional encoder
Defined in: conv_encoder.c

Output Variables:
*ptr_state   Initialized state variable of the encoder

```
void ctm_receiver(fifo_state_t*  ptr_signal_fifo_state,
                  fifo_state_t*  ptr_output_char_fifo_state,
                  Bool*          ptr_early_muting_required,
                  rx_state_t*    rx_state);
```

Purpose: Runs the CTM Receiver for a block of (nominally) 160 samples.
                    Due to the internal synchronization, the number of processed
                    samples might vary between 156 and 164 samples. The input of
                    the samples and the output of the decoded characters is
                    handled via fifo buffers, which have to be initialized
                    externally before using this function (see fifo.h for
                    details).
Defined in: ctm_receiver.c

input/output variables
*ptr_signal_fifo_state   fifo state for the input samples
*ptr_output_char_fifo_state fifo state for the output characters
*ptr_early_muting_required returns whether the original audio signal must not
                            be forwarded. This is to guarantee that the
                            preamble or resync sequence is detected only by the
                            first CTM device, if several CTM devices are
                            cascaded subsequently.
rx_state   pointer to the variable containing the receiver
                            states

```
void ctm_transmitter(UShortint   ucsCode,
                     Shortint*   txToneVec,
                     tx_state_t* tx_state,
                     Shortint    *ptrNumBitsStillToModulate,
                     Bool  sineOutput);
```

Purpose: Runs the CTM Transmitter for a block of 160 output samples,
                    representing 8 gross bits.
                    The bits, which are modulated into tones, are taken from an
                    internal fifo buffer. If the fifo buffer is empty, zero-valued
                    samples are generated. The fifo buffer is filled with channel-
                    encoded and interleaved bits, which are generated internally
                    by coding the actual input character. With each call of this
                    function one or less input characters can be coded. If there
                    is no character to for transmission, one of the following
                    codes has be used:
                    - 0x0016 <IDLE>: indicates that there is no character to
                    transmit and that the transmitter should stay in idle mode, if
                    it is currently already in idle mode. If the transmitter is
                    NOT in idle mode, it might generate <IDLE> symbols in order to
                    keep an active burst running. The CTM burst is terminated if
                    five <IDLE> symbols have been generated consecutively.
                    - 0xFFFF: although there is no character to transmit, a CTM
                    burst is initiated in order to signal to the far-end side that
                    CTM is supported. The burst starts with the <IDLE> symbol and
                    will be continued with <IDLE> symbols if there are no regular
                    characters handed over during the next calls of this function.
                    The CTM burst is terminated if five <IDLE> symbols have been

```
                    transmitted consecutively.
                    In order to avoid an overflow of the internal fifo buffer, the
                    variable *ptrNumBitsStillToModulate should be checked before
                    calling this function.
Defined in:         ctm_transmitter.c


input variables:
ucsCode                         UCS code of the character or one of the code 0x0016
                                or 0xFFFF
sineOutput                      must be false in regular mode; if true, a pure sine
                                output signal is generated

output variables:
txToneVec                       output signal (vector  of 160 samples)
input/output variables:
tx_state                        pointer to the variable containing the transmitter
                                states
```

```
void diag_deinterleaver(Shortint *out,
                    Shortint *in,
                    Shortint num_valid_bits,
                    interleaver_state_t *intl_state);
```

```
Purpose:          Corresponding deinterleaver to diag_interleaver. An arbitrary
                  number of bits can be interleaved, depending of the length of
                  the vector "in". The vector "out", which must have the same
                  length than "in", contains the interleaved samples.
                  All states (memory etc.) of the interleaver are stored in the
                  variable *intl_state. Therefore, a pointer to this variable
                  must be handled to this function. This variable initially has
                  to be initialized by the function init_interleaver, which
                  offers also the possibility to specify the dimensions of the
                  deinterleaver matrix.
Defined in:       diag_deinterleaver.c
```

```
void diag_interleaver(Shortint *out,
                    Shortint *in,
                    Shortint  num_bits,
                    interleaver_state_t *intl_state);
```

```
Purpose:          Diagonal (chain) interleaver, based on block-by-block
                  processing. An arbitrary number of bits can be interleaved,
                  depending of the value num_bits. The vector "out", which must
                  have the same length than "in", contains the interleaved
                  samples.
                  All states (memory etc.) of the interleaver are stored in the
                  variable *intl_state. Therefore, a pointer to this variable
                  must be handled to this function. This variable initially has
                  to be initialized by the function init_interleaver(), which
                  offers also the possibility to specify the dimensions of the
                  interleaver matrix.
Defined in:       diag_interleaver.c
```

```
void diag_interleaver_flush(Shortint *out,
                            Shortint *num_bits,
                            interleaver_state_t *intl_state);
```

Purpose: Execution of the diagonal (chain) interleaver without writing
                 in new samples. The number of calculated output samples is
                 returned via the value *num_bits.
Defined in: diag_interleaver.c

```
void generate_resync_sequence(Shortint *sequence);
```

Purpose: Generation of the sequence for resynchronization. The length
                 of the sequence is defined by the global constant
                 RESYNC_SEQ_LENGTH. The vector sequence must be allocated
                 accordingly before calling this function.
Defined in: wait_for_sync.c

```
void generate_scrambling_sequence(Shortint *sequence, Shortint length);
```

Purpose: Generation of the sequence used for scrambling. The sequence
                 consists of 0 and 1 elements. The sequence is stored into the
                 vector *sequence and the length of the sequence is specified
                 by the variable length.
Defined in: init_interleaver.c

```
void init_baudot_tonedemod(baudot_tonedemod_state_t* state);
```

Purpose: Initialization of the demodulator for Baudot Tones
Defined in: baudot_functions.c

Input/Output Variables:
state   Pointer to the initialized state variable (must be
                              allocated before calling init_baudot_tonedemod()

```
void init_baudot_tonemod(baudot_tonemod_state_t* state);
```

Purpose: Initialization of the modulator for Baudot Tones
Defined in: baudot_functions.c

Input/Output Variables:
state   Pointer to the initialized state variable (must be
                              allocated before calling init_baudot_tonemod()

```
void init_deinterleaver(interleaver_state_t *intl_state,
                        Shortint B, Shortint D);
```

Purpose: Initialization of the deinterleaver.
Defined in: init_interleaver.c

```
void init_ctm_receiver(rx_state_t* rx_state);
```

Purpose: Initialization of the CTM Receiver.
Defined in: ctm_receiver.c

output variables:
rx_state   pointer to a variable of rx_state_t containing the
                            initialized states of the receiver

---

```
void init_ctm_transmitter(tx_state_t* tx_state);
```

Purpose: Initialization of the CTM Transmitter
Defined in: ctm_transmitter.c

input/output variables
tx_state   pointer to a variable of tx_state_t containing
                            initialized states of the transmitter

---

```
void init_interleaver(interleaver_state_t *intl_state,
                      Shortint B, Shortint D,

                      Shortint num_sync_lines1, Shortint num_sync_lines2);
```

Purpose: Function for initialization of diag_interleaver and
                  diag_deinterleaver, respectively. The dimensions of the
                  interleaver must be specified:
                  B = (horizontal) blocklength, D = (vertical distance)
                  According to this specifications, this function initializes a
                  variable of type interleaver_state_t.
                  Additionally, this function adds two types of sync information
                  to the bitstream. The first sync info is for the demodulator
                  and consists of a sequence of alternating bits so that the
                  tones produced by the modulator are not the same all the time.
                  This is essential for the demodulator to find the transitions
                  between adjacent bits. The bits for this demodulator
                  synchronization simply precede the bitstream.
                  The second sync info is for synchronizing the deinterleaver
                  and of a m-sequence with excellent autocorrelation properties.
                  These bits are positioned at the locations of the dummy bits,
                  which are not used by the interleaver. In addition, even more
                  bits for this can be spent by inserting additional sync bits,
                  which precede the interleaver's bitstream. This is indicated
                  by choosing num_sync_lines2>0.
Defined in: init_interleaver.c

```
void init_tonedemod(demod_state_t *demod_state);
```

Purpose:	Initialization of one instance of the Tone Demodulator. The argument must contain a pointer to a variable of type demod_state_t, which contains all the memory of the tone demodulator. Each instance of tonedemod must have its own variable.
Defined In:	tonedemod.c

```
void init_wait_for_sync(wait_for_sync_state_t *ptr_wait_state,
               interleaver_state_t intl_state);
```

Purpose:	Initialization of the synchronization detector. The dimensions of the corresponding interleaver at the TX side must be specified by the variables B, D, and num_sync_lines2.
Defined In:	wait_for_sync.c

Input Variables:

B	(horizontal) blocklength
D	(vertical) interlace factor
num_Sync_line2	number of interleaver lines with additional sync bits (see description of init_interleaver())

Output Variables:

ptr_wait_state	pointer to the state variable of the sync detector

---

```
int main(int argc, const char** argv)
```

Purpose:	main function of the signal adaptation Module
Defined in:	adaptation_switch.c

```
Bool mutingRequired(Shortint  actualIndex,
          Shortint *mute_positions,
          Shortint  length_mute_positions);
```

Purpose:	Determines whether the actual bit has to be muted, i.e. whether it is contained in the vector mute_positions.
Defined in:	init_interleaver.c

```
void m_sequence(Shortint *sequence, Shortint length);
```

Purpose:	Calculates one period of an m-sequence (binary pseudo noise). The sequence is stored in the vector sequence, which must have a of $(2^r)-1$, where r is an integer number between 2 and 10. Therefore, with this release of m_sequence, sequences of length 3, 7, 15, 31, 63, 127, 255, 511, or 1023 can be generated. The resulting sequence is bipolar, i.e. it has values -1 and +1.
Defined in:	m_sequence.c

```
void polynomials(Shortint rate, Shortint k,
                 Shortint* polya, Shortint* polyb,
                 Shortint* polyc, Shortint* polyd);
```

Purpose:	Returns the polynomials for the convolutional encoder and the Viterbi decoder for various rates and constraint lengths. The following parameters are supported: rate = {2, 3, or 4} k = {3, 4, 5, 6, 7, 8, 9}
Defined in:	conv_poly.c

Input Variables:
rate	Rate of the convolutional encoder (2, 3, or 4)
k	Constraint length (length of the impulse response of the encoder)

Output Variables:
poly_a	Vector with polynomials #1
poly_b	Vector with polynomials #2
poly_c	Vector with polynomials #3 (only if rate > 2)
poly_d	Vector with polynomials #4 (only if rate > 3)

```
void reinit_deinterleaver(interleaver_state_t *intl_state);
```

Purpose:	Re-Initialization of the deinterleaver.
Defined in:	init_interleaver.c

```
void reinit_interleaver(interleaver_state_t *intl_state);
```

Purpose:	Re-initialization of the deinterleaver
Defined in:	init_interleaver.c

```
void reinit_wait_for_sync(wait_for_sync_state_t *ptr_wait_state);
```

Purpose:	Reinitialization of synchronization detector. This function is used in case that a burst has been finished and the transmitter has switched into idle mode. After calling reinit_wait_for_sync(), the function wait_for_sync() inhibits the transmission of the demodulated bits to the deinterleaver, until the next synchronization sequence can be detected.
Defined In:	wait_for_sync.c

```
void shift_deinterleaver(Shortint shift,
                         Shortint *insert_bits,

                         interleaver_state_t *ptr_state);
```

```
Purpose:            Shift of the deinterleaver buffer by <shift> samples.
                    shift>0  -> shift to the right
                    shift<0  -> shift to the left
                    The elements from <insert_bits> are inserted into the
                    resulting space. The vector <insert_bits> must have at least
                    abs(shift) elements.
Defined in:         diag_deinterleaver.c
```

```
Shortint sin_fip(Shortint phase_value);
```

```
Purpose:            Fixed Point sine function, returns the following value:
                    sin_fip(phase_value)
                      = round(32767*sin(2*pi*50/8000*phase_value))
                    phase_value must be within the range [0...159]. This function
                    can be used for calculating sine waveforms of frequencies that
                    are integer-multiples of 50 Hz
Defined in:         sin_fip.c
```

```
void tonedemod(Shortint *bits_out,
               Shortint *rx_tone_vec,
               Shortint num_in_samples,
               Shortint *ptr_sampling_correction,
               demod_state_t *demod_state);
```

```
Purpose:            Tone Demodulator for the CTM using one out of four tones for
                    coding two bits in parallel within a frame of 40 samples (5
                    ms).
                    The function has to be called for every frame of 40 samples of
                    the received tone sequence. However, in order to track a
                    non-ideal of the transmitter's and the receiver's clock
                    frequencies, one frame might be shorter (only 39 samples) or
                    longer (41 samples). The length of the following frame is
                    indicated by the variable *sampling_correction, which is
                    calculated and returned by this function.
```

```
Defined in:         tonedemod.c
```

```
input variables:
```

```
bits_out                    contains the 39, 40 or 41 actual samples of the
                            received tones; the bits are soft bits, i.e. they
                            are in the range between -1.0 and 1.0, where the
                            magnitude serves as reliability information
num_in_samples              number of valid samples in bits_out
```

```
output variables:
bits_out                    contains the two actual decoded soft bits
sampling_correction         is either -1, 0, or 1 and indicates whether the
                            next frame shall contain 39, 40, or 41 samples.
demod_state                 contains all the memory of tonedemod. Must be
                            initialized using the function init_tonedemod()
```

```
void tonemod(Shortint    *tones_out,
          Shortint    *bits_in,
```

```
        Shortint      num_samples_tones_out,
        Shortint      num_bits_in,
    mod_state_t *mod_state);
```

Purpose: Modulator for the CTM. The input vector bits_in must contain
                  the bits that have to be transmitted. The length of bits_in
                  must be even because always two bits are coded in parallel.
                  Bits are either unipolar (i.e. {0, 1}) or bipolar (i.e. {-1,
                  +1}). The length of the output vector tones_out must be 20
                  times longer than the length of bits_in, since each pair of
                  two bits is coded within a frame of 40 audio samples.

Defined In: tonemod.c

```
void transformUCS2UTF(UShortint      ucsCode,
                 fifo_state_t*  ptr_octet_fifo_state);
```

Purpose: Transformation from UCS code into UTF-8. UTF-8 is a sequence
                  consisting of 1, 2, 3, or 5 octets (bytes). See ISO/IEC
                  10646-1 Annex G.
                  This routine only handles UCS codes in the range 0...0xFF
                  since that is all that is required for the demonstration of
                  Baudot support.

Defined In: ucs_functions.c

Input Variables:
ucsCode   UCS code index

Output Variables:
ptr_octet_fifo_state   pointer to the output fifo state buffer for the
                            UTF-8 octets.

```
Bool transformUTF2UCS(UShortint      *ptr_ucsCode,
                 fifo_state_t*  ptr_octet_fifo_state)
```

Purpose: Transformation from UTF-8 into UCS code.

                  This routine only handles UTF-8 sequences consisting of one or
                  two octets (corresponding to UCS codes in the range 0...0xFF)
                  since that is all that is required for the demonstration of
                  Baudot support.

Defined In: ucs_functions.c

Input/Output Variables:
ptr_octet_fifo_state   pointer to the input fifo state buffer for the
                            UTF-8 octets.

Output Variables:
*ptr_ucsCode   UCS code index

```
Return Value:
true,                        if conversion was successful
false,                       if the input fifo buffer didn"t contain enough
                             octets for a conversion into UCS code. The output
                             variable *ptr_ucsCode doesn"t contain a value in
                             this case.
```

```
void viterbi_exec(Shortint*  inputword, Shortint  length_input,
                  Shortint*  out,        Shortint* num_valid_out_bits,
                  viterbi_t* viterbi_state);
```

```
Purpose:          Execution of the Viterbi decoder
Defined in:       viterbi.c

Input Variables:
inputword                    Vector with gross bits
length_input                 Number of valid gross bits in vector inputword.
                             length_input must be an integer multiple of
                             CHC_RATE.

Output variables:
out                          Vector with the decoded net bits. The net bits are
                             either 0 or 1.
*num_valid_out_bits          Number of valid bits in vector out.

Input/output variables:
*viterbi state               state variable of the decoder
```

```
void viterbi_init(viterbi_t* viterbi_state);
```

```
Purpose:          Initialization of the Viterbi decoder
Defined in:       viterbi.c

Output Variables:
*viterbi_state               Initialized state variable of the decoder
```

```
void viterbi_reinit(viterbi_t* viterbi_state);
```

```
Purpose:          Re-Initialization of the Viterbi decoder. This function should
                  be used for re-setting a Viterbi decoder that has already been
                  initialized. In contrast to init_viterbi(), this reinit
                  function does not calculate the values of all members of
                  viterbi_state that do not change during the execution of the
                  Viterbi algorithm.
Defined in:       viterbi.c

Output Variables:
*viterbi_state               Initialized state variable of the decoder
```

```
Bool wait_for_sync(Shortint *out_bits,
                   Shortint *in_bits,
```

```
                    Shortint   num_in_bits,
                    Shortint   num_received_idle_symbols,
                    Shortint   *ptr_num_valid_out_bits,
                    Shortint   *ptr_wait_interval,
                    Shortint   *ptr_resync_detected,
                    Bool       *ptr_early_muting_required,
                    wait_for_sync_state_t *ptr_wait_state);
```

Purpose: This function shall be inserted between the demodulator and
                  the deinterleaver. The function searches the synchronization
                  bitstream and cuts all received heading bits. As long as no
                  sync is found, this function returns
                  *ptr_num_valid_out_bits=0 so that the main program is able to
                  skip the deinterleaver as long as no valid bits are available.
                  If the sync info is found, the complete internal shift
                  register is copied to out_bits so that wait_for_sync can be
                  transparent and causes no delay for future calls.
                  *ptr_wait_interval returns a value of 0 after such a
                  synchronization indicating that this was a regular
                  synchronization.

                  Regularly, the initial preamble of each burst is used as sync
                  info. In addition, the resynchronization sequences, which
                  occur periodically during a running burst, are used as "back-
                  up" synchronization in order to avoid loosing all characters
                  of a burst, if the preamble was not detected.

                  If the receiver is already synchronized on a running burst and
                  the resynchronization sequence is detected,
                  *ptr_resync_detected returns a non-negative value in the range
                  0...num_in_bits-1 indicating at which bit the
                  resynchronization sequence has been detected. If no
                  resynchronization has been detected, *ptr_resync_detected is -
                  1. If the receiver is NOT synchronized and the
                  resynchronization sequence is detected, the resynchronization
                  sequence is used as initial synchronization.
                  *ptr_wait_interval returns a value of 32 in this case due to
                  the different alignments of the synchronizations based on the
                  preamble or the resynchronization sequence, respectively.

                  In order to carry all bits, the minimum length of out_bits
                  must be
                  in_bits.size()-1 + ptr_wait_state->shift_reg_length

Defined In: wait_for_sync.c

InputVariables:
in_bits   Vector with bits from the demodulator. The vector's
                           length can be arbitrarily chosen, i.e. according to
                           the block length of the signal processing of the
                           main program.
num_in_bits   length of vector in_bits

Output Variables:
num_received_idle_symbols Number if idle symbols received coherently
out_bits   Vector with bits for the deinterleaver. The number
                           of the valid bits is indicated by
                           *ptr_num_valid_out_bits.
*ptr_num_valid_out_bits   returns the number of valid output bits
*ptr_wait_interval   returns either 0 or 32
*ptr_resync_detected   returns a value -1, 0,...num_in_bits

```
*ptr_early_muting_required    returns whether the original audio signal must not
                              be forwarded. This is to guarantee that only the
                              first CTM device will detect the preamble or resync
                              sequence, if several CTM devices are cascaded
                              subsequently.
Input/Output Variables:
ptr_wait_state state information. This variable must be initialized with
                              init_wait_for_sync().
```

Annex A (informative):
Change history

<table>
<tr><td colspan="8" align="center">**Change history**</td></tr>
<tr><td>**Date**</td><td>**TSG SA#**</td><td>**TSG Doc.**</td><td>**CR**</td><td>**Rev**</td><td>**Subject/Comment**</td><td>**Old**</td><td>**New**</td></tr>
<tr><td>12-2000</td><td>10</td><td>SP-000570</td><td></td><td></td><td>Specification approved for Release 4</td><td></td><td>4.0.0</td></tr>
<tr><td>03-2001</td><td>11</td><td>SP-010108</td><td>001</td><td></td><td>Bug fix in source code of the CTM receiver</td><td>4.0.0</td><td>5.0.0</td></tr>
<tr><td>05-2001</td><td></td><td></td><td></td><td></td><td>Correct source code CTM attached</td><td>5.0.0</td><td>5.0.1</td></tr>
<tr><td>07-2004</td><td></td><td></td><td></td><td></td><td>Removed copyright terms and conditions in the source code CTM attached</td><td>5.0.1</td><td>5.0.2</td></tr>
<tr><td>12-2004</td><td>26</td><td></td><td></td><td></td><td>Version for Release 6</td><td>5.0.2</td><td>6.0.0</td></tr>
<tr><td>06-2007</td><td>36</td><td></td><td></td><td></td><td>Version for Release 7</td><td>6.0.0</td><td>7.0.0</td></tr>
<tr><td>03-2008</td><td>39</td><td>SP-080006</td><td>002</td><td>1</td><td>Bug fix to baudot_tonemod function in baudot_functions.c</td><td>7.0.0</td><td>7.1.0</td></tr>
<tr><td>12-2008</td><td>42</td><td></td><td></td><td></td><td>Version for Release 8</td><td>7.1.0</td><td>8.0.0</td></tr>
<tr><td>12-2009</td><td>46</td><td></td><td></td><td></td><td>Version for Release 9</td><td>8.0.0</td><td>9.0.0</td></tr>
<tr><td>03-2011</td><td>51</td><td></td><td></td><td></td><td>Version for Release 10</td><td>9.0.0</td><td>10.0.0</td></tr>
<tr><td>09-2012</td><td>57</td><td></td><td></td><td></td><td>Version for Release 11</td><td>10.0.0</td><td>11.0.0</td></tr>
</table>

History

Document history		
V11.0.0	October 2012	Publication