ETSI TS 126 268 V8.1.0 (2009-06)

Technical Specification

Digital cellular telecommunications system (Phase 2+);
Universal Mobile Telecommunications System (UMTS);
eCall data transfer;
In-band modem solution;
ANSI-C reference code
(3GPP TS 26.268 version 8.1.0 Release 8)

Reference
RTS/TSGS-0426268v810

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

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://webapp.etsi.org/IPR/home.asp).

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

Intellectual Property Rights 2
Foreword 2
Foreword 4
1 Scope 5
2 References 5
3 Abbreviations 5
4 C code structure 5
4.1 Contents of the C source code 6
4.2 Program execution 7
4.3 Variables, constants and tables 8
4.3.1 Description of constants used in the C-code 8
4.3.2 Type Definitions 9
4.3.3 Description of fixed tables used in the C-code 12
4.3.4 Static variables used in the C-code 13
4.4 Functions of the C Code 13
4.4.1 Interface functions 14
4.4.2 IVS transmitter functions 15
4.4.3 PSAP receiver functions 17
4.4.4 PSAP transmitter functions 19
4.4.5 IVS receiver functions 20
4.4.6 Synchronization functions (IVS and PSAP) 20
4.4.7 Other utility functions (IVS and PSAP) 21
Annex A (informative): Change history 22
History 23

Foreword

The present document has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the 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 TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG 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 document.

1 Scope

The present document contains an electronic copy of the ANSI-C code for the eCall in-band modem solution for reliable transmission of MSD data from IVS to PSAP via the speech channel of cellular networks. The ANSI-C code is necessary for a bit exact implementation of the IVS modem and PSAP modem described in 3GPP TS 26.267 [1].

2 References

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

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TS 26.267: "eCall Data Transfer; In-band modem solution; General description".

See also the references in 3GPP TS 26.267 [1].

3 Abbreviations

For the purpose of the present document, the following abbreviations apply:

ACK	ACKnowledgement
ANSI	American National Standards Institute
CRC	Cyclic Redundancy Check
FEC	Forward Error Correction
GSM	Global System for Mobile communications
HARQ	Hybrid Automatic Repeat-reQuest
I/O	Input/Output
IVS	In-Vehicle System
MSD	Minimum Set of Data
NACK	Negative ACKnowledgement
PCM	Pulse Code Modulation
PSAP	Public Safety Answering Point
RAM	Random Access Memory
ROM	Read Only Memory
RX	Receive
TX	Transmit

4 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 the present document.

The C code has been verified on the following systems:

- Windows XP SP2 and Microsoft Visual Studio V8.0;
- Linux (Suse Linux) using the gcc v3.4.2 and v4.1.2 compilers.

4.1 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.

Further explanation on the files is given in the `readme.txt` file, which is reproduced in part here:

```
Package Contents
----------------

folder 'ecall':
    Contains the complete eCall ANSI C fixed-point reference source code.

    modem_ivs.c     : top-level modem implementation for IVS
    modem_psap.c    : top-level modem implementation for PSAP

    modemx.h        : header file for both modem_ivs.c and modem_psap.c

    ecall_defines.h : compile time options and preprocessor constants

    ecall_fec.h     : header file FEC encoder and decoder
    ecall_modem.h   : header file modulator and demodulator
    ecall_sync.h    : header file synchronization
    ecall_rom.h     : header file ROM data

    ecall_fec.c     : FEC encoder and decoder
    ecall_modem.c   : modulator and demodulator
    ecall_sync.c    : synchronization
    ecall_rom.c     : ROM data


folder 'test_setup':
    Contains the eCall software simulation framework, to be compiled
    and run on MS Windows systems.

folder 'test_vec':
    Contains binary PCM data (104 files) and receiver/transmitter port logs
    in ASCII format (52 files) to test the eCall IVS and PSAP modems.

    The PCM format is 16 bit signed, little endian, at 8 kHz sampling rate.
    The data files reflect 26 test cases and were generated from the eCall
    simulation framework.

    campaign_short.txt  : configuration file for the 26 test cases

    pcmdlout<index>.pcm : output PCM data of DL vocoder = input to IVS
    pcmulout<index>.pcm : output PCM data of UL vocoder = input to PSAP

    pcmdlin<index>.pcm  : test vectors for PSAP modem output
    pcmulin<index>.pcm  : test vectors for IVS modem output


    portpsap<index>.txt : test vectors for PSAP port logs
    portivs<index>.txt  : test vectors for IVS port logs


folder 'TEST_SETUP':
    Contains a test setup for an eCall transmission.

standalone.c
    main() wrapper to run the IVS or PSAP modem on prestored PCM files or
    receiver/transmitter port logs. To get a list of command-line options,
    invoke the corresponding executable with option '-h' (help).

standalone.h
    header file for standalone.c


Makefile.win
```

```
    Microsoft Visual Studio 2005/2008 Makefile
    Builds 'standalone.exe' from standalone.c and the eCall sources,
    build options are RELEASE and DEBUG.


Makefile.glx
    GNU Linux Makefile using gcc
    Builds 'standalone' from standalone.c and the eCall sources,
    build options are RELEASE and DEBUG.

verify.bat
    Windows batch file
    Runs 'standalone.exe' in six different modem modes on the 26 test cases
    contained in folder 'test_vec' and performs a test vector comparison to
    the respective output PCM and port log data.

verify.sh
    Linux shell script
    Runs 'standalone' in mode '-m ivs' and '-m psap' on 26 test cases
    (folder 'pcm') and performs a test vector comparison to the respective
    modem output PCM data.
```

4.2 Program execution

An explanation on code compilation and execution is given in the `readme.txt` file, which is reproduced in part here:

```
Getting Started
---------------

3GPP TS 26.268 provides the eCall modem source code, a software simulation
framework, and a standalone wrapper that allows to run the IVS or PSAP modem
on prestored reference data.

Five functions represent the eCall modem interface, which in turn invoke the
respective receiver (Rx) and transmitter (Tx) implementation of each modem:

    * PsapSendStart
    * PsapReset      -> invokes:  PsapRxReset,   PsapTxReset
    * PsapProcess    -> invokes:  PsapRxProcess, PsapTxProcess
    * IvsReset       -> invokes:  IvsRxReset,    IvsTxReset
    * IvsProcess     -> invokes:  IvsRxProcess,  IvsTxProcess

The external application must in addition implement the callback function
PsapReceiveMsd, which the PSAP modem will call once the MSD was successfully
received. See standalone.c for an example.

For a real-time simulation over 3GPP FR and AMR vocoders and to log PCM data
as input to the standalone wrapper, the eCall sources have to be integrated
into a simulation framework (test setup); the one used in the 3GPP selection tests is
attached in the subfolder TEST_SETUP. The basic integration steps are briefly described
below.

In order to compile and run the eCall modem code, follow the instructions
given below. For code testing, two batch files have been provided:

    * verify.bat : MS Windows systems
    * verify.sh  : Linux systems

For each of the 26 test cases of campaign_short.txt in folder 'test_vec',
they run the standalone wrapper in six different modem modes (three IVS and
three PSAP modes). The resulting PCM and port log files in folder 'out' are
finally compared to the test vectors in folder 'test_vec'.

In modes 'psap' and 'psaprx', you should see the message 'MSD received!' on
completion of each test case.
```

```
Code Compilation
----------------

MS Windows systems
    To build standalone.exe from standalone.c and the eCall sources, start
    with opening a new project in Visual Studio 2005/2008.

    Choose File -> New -> 'Project from Existing Code' and follow the
    instructions of the 'Create Project from Existing Code Files Wizard'.
    Configuration:

    * Type of project: Visual C++
    * Specify the folder location of standalone.c and a project name
    * Button 'Next'
    * Select 'Use external build system'
    * For Debug and Release configuration, specify

    Build command line:  nmake -f Makefile.win
    Clean command line:  nmake -f Makefile.win clean

    Build the project with shorthand key 'F7' or from the menu.
    The source code should compile without any errors or warnings.
    Run 'verify.bat' to verify the executable against the test vectors.


GNU Linux systems
    Compilation under Linux has been tested with

    * GNU Make version 3.81
    * gcc version 4.1.3 and 4.2.4

    For building the executable 'standalone' and cleanup, use

    make -f Makefile.glx
    make -f Makefile.glx clean

    On the platforms tested, the code compiled without errors or warnings.
    Run 'verify.sh' to verify the executable against the test vectors.


Simulation Framework
--------------------

See LICENSE.TXT and README.TXT in folder TEST_SETUP for terms of usage!

Note that this simulation framework has to be compiled and run on MS Windows
operating systems, as Windows specific API functions are used and the FR and
AMR vocoders are attached to the framework in form of Windows executables.

To attach the eCall sources to the framework, copy the 'ecall' folder into
the 'c' folder of directory TEST_SETUP. Compile and link the *.c files under
subfolder 'ecall' by adding their corresponding object files to the list of
makefile targets. Note that modem_ivs.c and modem_psap.c replace the code
template modem_demo.c.
```

4.3 Variables, constants and tables

4.3.1 Description of constants used in the C-code

This clause contains a listing of all global constants defined in `ecall_defines.h`., together with some explanatory comments.

Constant	Value	Description

```
#define MAX(a,b)                ((a)>(b) ? (a) : (b))
#define MIN(a,b)                ((a)<(b) ? (a) : (b))
#define ABS(a)                  ((a)<0 ? (-a) : (a))
#define SIGN(a)                 ((a)<0 ? (-1) : (1))

#define PCM_LENGTH              160             length of PCM frame
#define MSD_MAX_LENGTH          140             length of MSD message (bytes)

/* Synchronization */
#define SYNC_BADCHECKS          (8)             IVS subsequent bad checks
#define SYNC_IDXLEN             (75)            sync index length
#define SYNC_THRESHOLD          (10e6)          sync threshold

#define LOCK_START              (3)             number of START to lock sync
#define FAIL_RESTART            (3)             number of START messages to restart

#define NRF_WAKEUP              (3)             number of wakeup frames
#define NRF_SYNC                (13)            length of sync in frames
#define NRF_OBSERVE             (10)            number of frames the PSAP checks for a
                                                  better sync after detecting a preamble

#define PNSEQ_OSF               (22)            "oversampling" rate of PN sequence
#define PEAK_DIST_PP            (30*PNSEQ_OSF)  distance outer positive peaks
#define PEAK_DIST_NN            (54*PNSEQ_OSF)  distance negative peaks
#define PEAK_DIST_PN            (12*PNSEQ_OSF)  distance positive to negative

/* Uplink/Downlink format */
#define ARQ_MAX                 (8)             number of redundancy versions
#define NRB_TAIL                (3)             number of encoder tail bits
#define NRB_CRC                 (28)            order of CRC polynomial

#define NRB_INFO                (8*MSD_MAX_LENGTH)
#define NRB_INFO_CRC            (8*MSD_MAX_LENGTH + NRB_CRC)
#define NRB_CODE_ARQ            (1380)
#define NRB_CODE_BUFFER         (3*(8*MSD_MAX_LENGTH + NRB_CRC) + 4*NRB_TAIL)

#define NRF_DLDATA              (3)             DL data part
#define NRF_DLMUTE1             (2)             DL 1st muting (after sync)
#define NRF_DLMUTE2             (2)             DL 2nd muting (after data)
#define NRF_DLCHUNK             (NRF_SYNC + NRF_DLMUTE1 + NRF_DLDATA + NRF_DLMUTE2)

/* IVS/PSAP processing */
#define NRF_MEMIVS              (5)             buffer size in frames (IVS)
#define NRF_MEMPSAP             (2)             buffer size in frames (PSAP)
#define NRS_MEMSYNC             (820)           memory size in samples (SYNC)

#define IVS_THRESHOLD           (40000)         threshold feedback messages
#define IVS_GOSTART             (6)             threshold for unreliable START
#define IVS_TXFAST              (10)            fast modulator mode NACK condition
#define IVS_TXINC               (87)            sample increment at restart

#define PSAP_MSDACK             (5)             number of PSAP ACK messages

#define FEC_VAR                 (30206)         variance: 1/4550000 in Q37
#define FEC_MEAN                (0xB9999A)      mean: 5.8 in Q21
#define FEC_ITERATIONS          (8)             number of decoder iterations
#define FEC_STATES              (8)             number of decoder states

#define IntLLR                  Int16           size of soft bit buffer variables
#define LLR_MAX                 ((Int32)(0x7fff-1))
#define LOGEXP_RES              (401)           resolution of LOGEXP table
#define LOGEXP_DELTA            (-6)            determines internal Q-factor
#define LOGEXP_QIN              (8)             input Q-factor of LLR values
```

4.3.2 Type Definitions

The following type definitions have been used, which are defined in ecall_defines.h, ecall_modem.h, ecall_sync.h, and modemx.h:

Definition	Description

```
typedef enum { False, True } Bool;      boolean variable

typedef signed char        Int8;        8 bit signed variable
typedef signed short int   Int16;       16 bit signed variable
typedef signed int         Int32;       32 bit signed variable

typedef unsigned char      Ord1;        binary symbol
typedef unsigned char      Ord8;        8 bit unsigned variable
typedef unsigned short int Ord16;       16 bit unsigned variable
typedef unsigned int       Ord32;       32 bit unsigned variable

typedef enum {
  ModUndef,
  Mod3bit4smp,
  Mod3bit8smp
} ModType;                              modulator type for uplink transmission

typedef struct {
  ModType type;                         identifies modulator type

  Int16 bpsym;                          bits per symbol
  Int16 spmf;                           samples per modulation frame
  Int16 mfpf;                           modulation frames per frame = PCM_LENGTH/spmf
  Int16 decpos1;                        position 1st decoding trial
  Int16 decpos2;                        position 2nd decoding trial
  Int16 wutperiod;                      wakeup tone period in samples
  Int16 nfmute1;                        number of muting frames 1st interval
  Int16 nfmute4;                        number of muting frames 4th interval
  Int16 nfmuteall;                      number of muting frames total
  Int16 nfdata;                         number of data frames = NRB_CODE_ARQ/(mfpf*bpsym)

  const Int16 *ulPulse;
  const Int16 *ulPulseMatch;
  const Int16 *mgTable;
  const Int16 *wakeupSin;
  const Int16 *wakeupCos;
} ModState;                             modulator state for uplink transmission

typedef struct {
  Int32 *state;
  Int32 *wakeupState;                   memory for wakeup tone detector

  Int32 amplitude[3];                   amplitudes (average, maximum, memory)
  Int16 corrIndex[4];                   position of sync check
  Int32 checkMem[4];                    correlation value memory of sync check
  Int16 peakPos[4];                     position of sync peaks within feedback message

  Int16 index;                          frame reference for sync evaluation
  Int16 offset;                         frame offset
  Int16 delay;                          synchronization delay (position)
  Int16 tempDelay;                      temporary delay in two-stage peak evaluation
  Int16 prevDelay;                      previous sync delay
  Int16 trials;                         number of sync trials
  Int16 npeaks;                         number of sync peaks detected
  Int16 events;                         number of subsequent equal sync events

  Bool  flag;                           indicates successful sync
  Bool  checkOk;                        indicates successful sync check
} SyncState;                            state of synchronization functions
typedef enum {
  DlMsgStart,
  DlMsgNack,
  DlMsgAck,
  DlMsgIdle,
  DlTriggerReset,
  DlNoop

} DlData;                               downlink message identifiers
```

```
typedef enum {
  IvsIdle,
  IvsSendMsd
} IvsState;                          IVS state identifiers

typedef struct {
  SyncState sync;                    IVS sync struct

  Bool  dlRead;                      sync indication
  Int16 dlIndex;                     downlink frame counter
  Int16 checkCnt;                    counter for subsequent sync check failures

  Int16 pcmBuffer[NRF_MEMIVS*PCM_LENGTH];
  Int32 syncBuffer[NRS_MEMSYNC];
} IvsRxData;

typedef struct {
  ModState mod;                      IVS modulator struct

  Bool  dlSyncLock;                  RX->TX PORT: downlink sync lock trigger
  Int16 dlData;                      RX->TX PORT: downlink message symbol
  Int32 dlMetric;                    RX->TX PORT: downlink metric

  Int16 state;                       IVS state
  Int16 stateCnt[4];                 state counters
  Int16 stateCntNack;                global counter for NACK messages
  Int16 startIgnored;                counter for unreliable START messages
  Bool  startPending;                indicates pending START message

  Int16 delay;                       transmit offset in samples
  Int16 rv;                          redundancy version
  Int16 ulN;                         uplink number of frames
  Int16 ulIndex;                     uplink frame counter

  Ord1  bitBuffer[NRB_CODE_BUFFER];
  Int16 delayBuffer[2*PCM_LENGTH];
} IvsTxData;

typedef struct {
  IvsRxData rx;                      IVS receiver struct
  IvsTxData tx;                      IVS transmitter struct
} IvsData;

typedef enum {
  PsapIdle,
  PsapStart,
  PsapNack,
  PsapAck,
  PsapTrigger,
} PsapState;                          PSAP state identifiers

typedef struct {
  ModState  mod;                     PSAP modulator struct
  SyncState sync;                    PSAP sync struct

  Int16 state;                       PSAP state
  Int16 rv;                          redundancy version
  Int16 ulN;                         uplink number of frames (without muting)
  Int16 ulIndex;                     uplink frame counter
  Int16 mgIndex;                     uplink position in muting gap table
  Int16 decTrials;                   decoding trails
  Int16 observeCnt;                  counter for frames after successful sync

  Int16 dlData;                      downlink message symbol
  Int16 dlIndex;                     donwlink frame counter
  Int16 dlMsgCnt;                    downlink message counter

  Ord8   *msd;                       MSD in byte representation
  Ord1   *msdBin;                    MSD in binary representation
```

```
  Int16  *pcmBuffer;                  sync buffer
  IntLLR *bitBuffer;                  soft bit buffer for decoding

  char buffer[0
    + sizeof(IntLLR)* NRB_CODE_ARQ
    + sizeof(Int16) * NRF_MEMPSAP*PCM_LENGTH
    + sizeof(Int32) * NRS_MEMSYNC
    + sizeof(Int32) * 2*(NRF_SYNC+1)];
} PsapRxData;

typedef struct {
  Int16 dlData;                       RX->TX PORT: downlink message symbol
  Int16 dlIndex;                      RX->TX PORT: donwlink frame counter
} PsapTxData;

typedef struct {
  PsapRxData rx;                      PSAP receiver struct
  PsapTxData tx;                      PSAP transmitter struct
  Int16 msgCounter;                    message counter
} PsapData;
```

4.3.3 Description of fixed tables used in the C-code

This clause contains a listing of all fixed tables (ROM) defined in `ecall_rom.c`.

Type/Constant	Dimension	Description
/* Synchronization */		
Int16 wakeupSin500	[16]	sine waveform at 500 Hz
Int16 wakeupCos500	[16]	cosine waveform at 500 Hz
Int16 wakeupSin800	[10]	sine waveform at 800 Hz
Int16 wakeupCos800	[10]	cosine waveform at 800 Hz
Int16 syncPulseForm	[5]	sync pulse
Int16 syncSequence	[15]	sync pulse sequence
Int16 syncIndexPreamble	[SYNC_IDXLEN]	sync pulse positions
Int16 syncFrame	[1600]	predefined synchronization signal
/* Uplink/Downlink format */		
Int16 indexBits	[24]	bit positions for turbo decoder
// fast modulator mode:		
Int16 m4smp_ulPulse	[16]	uplink waveform
Int16 m4smp_ulPulseMatch	[64]	matched filtered uplink waveform
Int16 m4smp_mgTable	[54]	table indicating muting gaps
// robust modulator mode:		
Int16 m8smp_ulPulse	[32]	uplink waveform
Int16 m8smp_ulPulseMatch	[128]	matched filtered uplink waveform
Int16 m8smp_mgTable	[104]	table indicating muting gaps
Int16 dlPcmData	[4][NRF_DLDATA*PCM_LENGTH]	downlink transmit signal
Int16 dlPcmDataMatch	[4][NRF_DLDATA*PCM_LENGTH]	DL MF signal
/* FEC encoder/decoder */		
Ord16 stateTransMat	[8][2]	FEC: state transitions
Ord16 stateTrans	[16]	FEC: state transitions
Ord16 revStateTransMat	[8][2]	FEC: reverse state transitions
Ord16 revStateTrans	[16]	FEC: reverse state transitions
Ord1 outputParityMat	[8][2]	FEC: output parity indicator
Ord1 outputParity	[16]	FEC: output parity indicator
Ord1 crcPolynomial	[NRB_CRC+1]	coefficients of CRC polynomial
Ord1 scramblingSeq	[NRB_INFO_CRC]	bit scrambling sequence
Ord16 interleaverSeq	[NRB_INFO_CRC]	interleaver sequence
Ord16 redVerIndex	[8][NRB_CODE_ARQ]	index vector for HARQ process
IntLLR logExpTable	[LOGEXP_RES]	lookup table (logExp function)

4.3.4 Static variables used in the C-code

This clause contains a listing of static variables (RAM) defined in source files.

Definition	Description
`IvsData ivs`	`IVS static memory`
`PsapData psap`	`PSAP static memory`
`WordLLR chCodedSoftBitBuffer[NRB_CODE_BUFFER]`	`soft bit buffer of turbo decoder`

4.4 Functions of the C Code

This clause contains the headers of the employed IVS and PSAP functions. They correspond to a large extent to the functional description of the IVS and PSAP provided in 3GPP TS 26.267 [1].

Figure 1 gives an overview of the most important functions and their hierarchical relation.

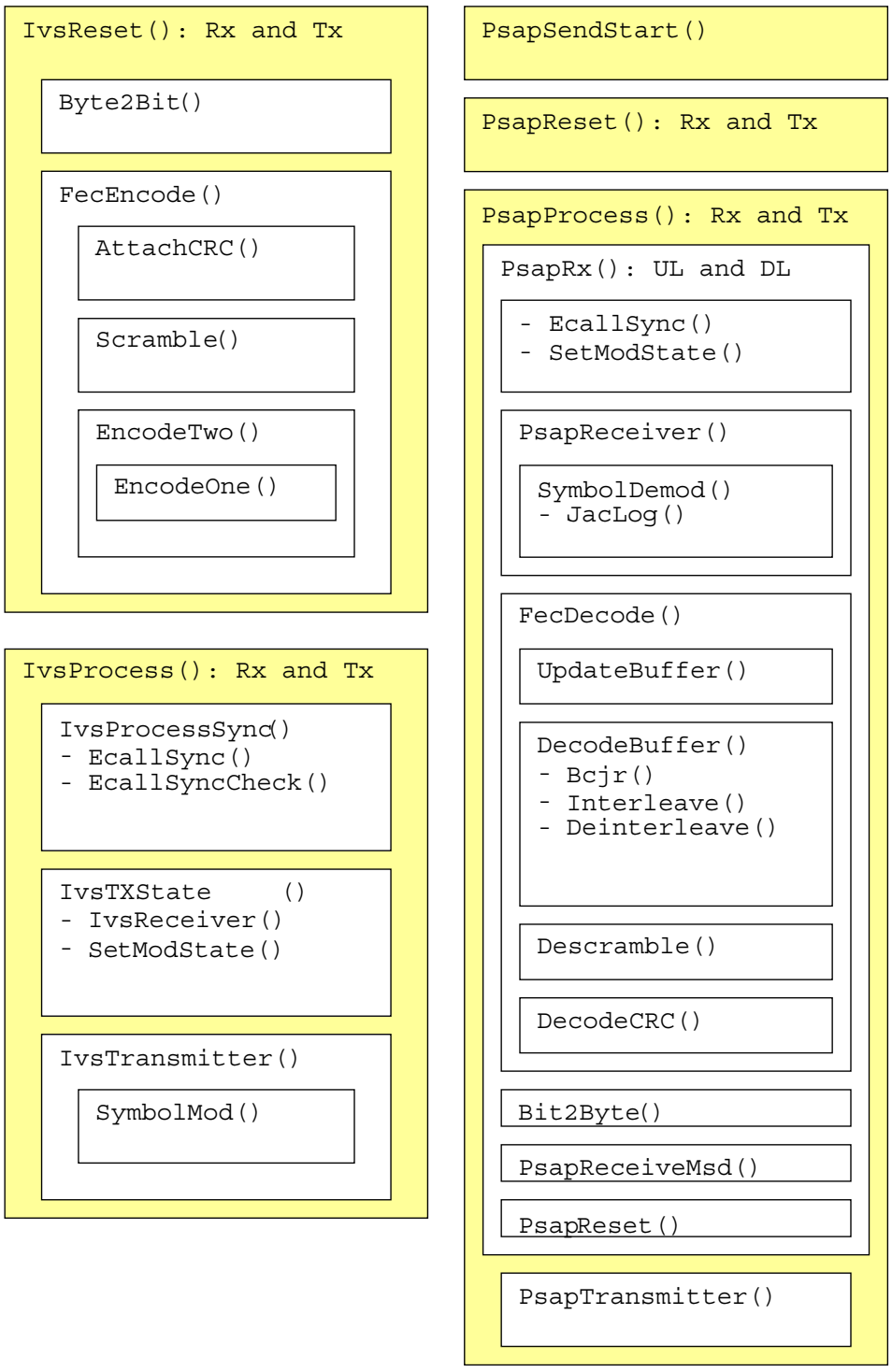

Figure 1: Hierarchical function overview

4.4.1 Interface functions

```
/*============================================================================*/
/* IVS  implementation: IvsReset                                              */
/*----------------------------------------------------------------------------*/
/* Description: Reset of IVS before the reception of a new MSD                */
/*                                                                            */
/* In:  const Ord8* msd     -> MSD to be transmitted                          */
/*      int         length  -> MSD length (equal to MSD_MAX_LENGTH)           */
/*----------------------------------------------------------------------------*/
void IvsReset(const Ord8 *msd, int length)
```

```
void IvsRxReset()
void IvsTxReset(const Ord8 *msd, int length)


/*============================================================================*/
/* IVS  implementation: IvsProcess                                            */
/*----------------------------------------------------------------------------*/
/* Description: IVS modem function that processes the PCM data                */
/*                                                                            */
/* InOut:  Int16* pcm  <-> input and output frame of 16bit PCM samples        */
/*----------------------------------------------------------------------------*/
void IvsProcess(Int16 *pcm)
void IvsRxProcess(const Int16 *pcm)
void IvsTxProcess(Int16 *pcm)


/*============================================================================*/
/* PSAP  implementation: PsapSendStart                                        */
/*----------------------------------------------------------------------------*/
/* Description: Initiates PSAP to trigger the transmission of an MSD          */
/*----------------------------------------------------------------------------*/
void PsapSendStart()


/*============================================================================*/
/* PSAP  implementation: PsapReset                                            */
/*----------------------------------------------------------------------------*/
/* Description: Reset of PSAP before the reception of a new MSD               */
/*----------------------------------------------------------------------------*/
void PsapReset()
void PsapRxReset()
void PsapTxReset()


/*============================================================================*/
/* PSAP  implementation: PsapProcess                                          */
/*----------------------------------------------------------------------------*/
/* Description: PSAP modem function that processes the PCM data               */
/*                                                                            */
/* InOut:  Int16* pcm  <-> input and output frame of 16bit PCM samples        */
/*----------------------------------------------------------------------------*/
void PsapProcess(Int16 *pcm)
void PsapRxProcess(const Int16 *pcm)
void PsapTxProcess(Int16 *pcm)
```

4.4.2 IVS transmitter functions

```
/*============================================================================*/
/* IVS FUNCTION: IvsTransmitter                                               */
/*----------------------------------------------------------------------------*/
/* Description: IVS transmitter function                                      */
/*                                                                            */
/* In:     const ModState* ms      -> modulator struct                        */
/*         const Ord1*     buffer  -> code bit buffer                         */
/*         Int16           rv      -> redundancy version                      */
/*         Int16           index   -> position within uplink frame            */
/* Out:    Int16*          pcm     <- output data                             */
/*----------------------------------------------------------------------------*/
void IvsTransmitter(const ModState *ms, const Ord1 *buffer, Int16 *pcm,
                    Int16 rv, Int16 index)


/*============================================================================*/
/* UTILITY FUNCTION: IvsTxState                                               */
/*----------------------------------------------------------------------------*/
/* Description: IVS state machine evaluating feedback messages                */
/*                                                                            */
/* In:  Int16 msg     -> new downlink message symbol                          */
```

```
/*      Int32 metric  -> downlink metric (if negative ignore symbol)          */
/*----------------------------------------------------------------------------*/
void IvsTxState(Int16 msg, Int32 metric)


/*============================================================================*/
/* IVS FUNCTION: SymbolMod                                                    */
/*----------------------------------------------------------------------------*/
/* Description: symbol modulator                                              */
/*                                                                            */
/* In:     const ModState* ms      -> modulator struct                        */
/*         Int16           symbol  -> symbol index                            */
/* Out:    Int16*          mPulse  <- modulated output sequence               */
/*----------------------------------------------------------------------------*/
void SymbolMod(const ModState *ms, Int16 symbol, Int16 *mPulse)


/*============================================================================*/
/* IVS FUNCTION: Byte2Bit                                                     */
/*----------------------------------------------------------------------------*/
/* Description: conversion byte vector to bit vector                          */
/*                                                                            */
/* In:     Ord8* in      -> vector of input bytes                             */
/*         Int16 length  -> length of input                                   */
/* Out:    Ord1* out     <- vector of output bits                             */
/*----------------------------------------------------------------------------*/
void Byte2Bit(const Ord8 *in, Ord1 *out, Int16 length)


/*============================================================================*/
/* ENCODER FUNCTION: FecEncode                                                */
/*----------------------------------------------------------------------------*/
/* Description: encoding of MSD                                               */
/*                                                                            */
/* InOut:  Ord1 *buffer  <-> takes info bits and returns coded bits           */
/*----------------------------------------------------------------------------*/
void FecEncode(Ord1 *buffer)


/*============================================================================*/
/* ENCODER FUNCTION: AttachCrc                                                */
/*----------------------------------------------------------------------------*/
/* Description: attaches CRC bits                                             */
/*                                                                            */
/* In:     const Ord1* infoBits     -> input information bits                 */
/* Out:    Ord1*       infoWithCrc  <- bits with CRC attached                 */
/*----------------------------------------------------------------------------*/
void AttachCrc(const Ord1 *infoBits, Ord1 *infoWithCrc)


/*============================================================================*/
/* ENCODER FUNCTION: Scramble                                                 */
/*----------------------------------------------------------------------------*/
/* Description: bit scrambling                                                */
/*                                                                            */
/* In:     const Ord1* in   -> non scrambled input bit sequence               */
/* Out:    Ord1*       out  <- scrambled output bit sequence                  */
/*----------------------------------------------------------------------------*/
void Scramble(const Ord1 *in, Ord1 *out)

/*============================================================================*/
/* ENCODER FUNCTION: EncodeTwo                                                */
/*----------------------------------------------------------------------------*/
/* Description: encoding of bit sequence                                      */
/*                                                                            */
/* InOut:  Ord1* codedBits  <-> scrambled bits to coded bits                  */
/*----------------------------------------------------------------------------*/
void EncodeTwo(Ord1 *codedBits)
```

```
/*============================================================================*/
/* ENCODER FUNCTION: EncodeOne                                                */
/*----------------------------------------------------------------------------*/
/* Description: convolutional encoding of each component                      */
/*                                                                            */
/* In:     Int16 encNr       -> component number                              */
/* InOut:  Ord1* codedBits  <-> bits to be encoded                            */
/*----------------------------------------------------------------------------*/
void EncodeOne(Ord1 *codedBits, Int16 encNr)
```

4.4.3 PSAP receiver functions

```
/*============================================================================*/
/* UTILITY FUNCTION: PsapRxUplink                                             */
/*----------------------------------------------------------------------------*/
/* Description: PSAP UL state machine, determines PSAP receiver operation     */
/*              according to the state                                        */
/*                                                                            */
/* In:  const Int16* pcm  -> input frame of 16bit PCM samples                 */
/*----------------------------------------------------------------------------*/
void PsapRxUplink(const Int16 *pcm)


/*============================================================================*/
/* UTILITY FUNCTION: PsapRxDownlink                                         */
/*----------------------------------------------------------------------------*/
/* Description: PSAP DL state machine, determines PSAP transmitter operation  */
/*              according to the state                                        */
/*----------------------------------------------------------------------------*/
void PsapRxDownlink()


/*============================================================================*/
/* PSAP FUNCTION: PsapReceiver                                                */
/*----------------------------------------------------------------------------*/
/* Description: PSAP receiver function (decoding is done outside)             */
/*                                                                            */
/* In:     const ModState* ms        -> modulator struct                      */
/*         const Int16*    pcm       -> input data for demodulation           */
/* Out:    IntLLR*         softBits  <- demodulated soft bit sequence         */
/*----------------------------------------------------------------------------*/
void PsapReceiver(const ModState *ms, const Int16 *pcm, IntLLR *softBits)


/*============================================================================*/
/* PSAP FUNCTION: SymbolDemod                                                 */
/*----------------------------------------------------------------------------*/
/* Description: symbol demodulator                                            */
/*                                                                            */
/* In:     const ModState* ms        -> modulator struct                      */
/*         const Int16*    mPulse    -> received pulse train                  */
/* Out:    IntLLR*         softBits  <- demodulated soft bit sequence         */
/*----------------------------------------------------------------------------*/
void SymbolDemod(const ModState *ms, const Int16 *mPulse, IntLLR *softBits)


/*============================================================================*/
/* PSAP FUNCTION: Bit2Byte                                                    */
/*----------------------------------------------------------------------------*/
/* Description: conversion bit vector to byte vector                          */
/*                                                                            */
/* In:     const Ord1* in      -> vector of input bits                        */
/*         Int16       length  -> length of output                            */
/* Out:    Ord8*       out     <- vector of output bytes                      */
/*----------------------------------------------------------------------------*/
void Bit2Byte(const Ord1 *in, Ord8 *out, Int16 length)
```

```
/*============================================================================*/
/* PSAP FUNCTION: MpyLacc                                                     */
/*----------------------------------------------------------------------------*/
/* Description: multiply 32bit number with 16bit number (32bit result)        */
/*                                                                            */
/* In:     Int32 var32  -> 32bit number                                       */
/*         Int16 var16  -> 16bit number                                       */
/* Return: Int32        <- result                                             */
/*----------------------------------------------------------------------------*/
Int32 MpyLacc(Int32 var32, Int16 var16)

/*============================================================================*/
/* DECODER FUNCTION: FecDecode                                                */
/*----------------------------------------------------------------------------*/
/* Description: decoding to find the MSD                                      */
/*                                                                            */
/* In:     const IntLLR* in   -> received soft bits                           */
/*         Int16         rv   -> redundancy version                           */
/* Out:    Ord1*         out  <- decoded MSD in binary representation         */
/* Return: Bool               <- result of CRC check                          */
/*----------------------------------------------------------------------------*/
Bool FecDecode(const IntLLR *in, Int16 rv, Ord1 *out)


/*============================================================================*/
/* DECODER FUNCTION: UpdateBuffer                                             */
/*----------------------------------------------------------------------------*/
/* Description: update channel LLR buffer with new soft bits                  */
/*                                                                            */
/* In:     const IntLLR* softInBits    -> received soft bits                  */
/*         Int16         rv            -> redundancy version                  */
/* InOut:  IntLLR*       chLLRbuffer  <-> decoder buffer                      */
/*----------------------------------------------------------------------------*/
void UpdateBuffer(IntLLR *chLLRbuffer, const IntLLR *softInBits, Int16 rv)


/*============================================================================*/
/* DECODER FUNCTION: DecodeBuffer                                             */
/*----------------------------------------------------------------------------*/
/* Description: decoding of LLR buffer                                        */
/*                                                                            */
/* In:     const IntLLR* syst1    -> RX systematic soft bits                  */
/*         const IntLLR* syst2    -> interleaved RX systematic tail bits      */
/*         const IntLLR* parity1  -> RX parity soft bits                      */
/*         const IntLLR* parity2  -> interleaved RX parity soft bits          */
/* Out:    Ord1*         decBits  <- decoded bits                             */
/*----------------------------------------------------------------------------*/
void DecodeBuffer(const IntLLR *syst1, const IntLLR *syst2,
                  const IntLLR *parity1, const IntLLR *parity2, Ord1 *decBits)


/*============================================================================*/
/* DECODER FUNCTION: Bcjr                                                     */
/*----------------------------------------------------------------------------*/
/* Description: BCJR algorithm                                                */
/*                                                                            */
/* In:     const IntLLR* parity      -> received parity soft bits             */
/* InOut:  IntLLR*       extrinsic  <-> extrinsic information                 */
/*----------------------------------------------------------------------------*/
void Bcjr(const IntLLR *parity, IntLLR *extrinsic)


/*============================================================================*/
/* DECODER FUNCTION: Interleave                                               */
/*----------------------------------------------------------------------------*/
/* Description: Turbo code interleaver                                        */
/*                                                                            */
/* In:     const IntLLR* in   -> input sequence                               */
```

```
/* Out:    IntLLR*       out  <- output sequence                                 */
/*-------------------------------------------------------------------------------*/
void Interleave(const IntLLR *in, IntLLR *out)


/*===============================================================================*/
/* DECODER FUNCTION: Deinterleave                                                */
/*-------------------------------------------------------------------------------*/
/* Description: Turbo code deinterleaver                                         */
/*                                                                               */
/* InOut:  IntLLR* inout  <-> input and deinterleaved output sequence            */
/*-------------------------------------------------------------------------------*/
void Deinterleave(IntLLR *inout)

/*===============================================================================*/
/* DECODER FUNCTION: Descramble                                                  */
/*-------------------------------------------------------------------------------*/
/* Description: descrambles decoded bits                                         */
/*                                                                               */
/* InOut:  Ord1* inout  <-> input and output bit sequence                        */
/*-------------------------------------------------------------------------------*/
void Descramble(Ord1 *inout)


/*===============================================================================*/
/* DECODER FUNCTION: DecodeCrc                                                   */
/*-------------------------------------------------------------------------------*/
/* Description: check CRC of decoded bits                                        */
/*                                                                               */
/* In:     const Ord1* codedBits  -> decoded bit sequence to be checked          */
/* Return: Bool                   <- result of CRC check                         */
/*-------------------------------------------------------------------------------*/
Bool DecodeCrc(const Ord1 *codedBits)


/*===============================================================================*/
/* DECODER FUNCTION: GammaQ                                                      */
/*-------------------------------------------------------------------------------*/
/* Description: compute gamma values for BCJR algorithm                          */
/*                                                                               */
/* In:     Int16         k          -> bit position                              */
/*         Int16         l          -> state                                     */
/*         const IntLLR* parity     -> received parity bits                      */
/*         const IntLLR* extrinsic  -> sum of extrinsic and systematic bits      */
/* Return: IntLLR                   <- value of gamma(k,l)                       */
/*-------------------------------------------------------------------------------*/
IntLLR GammaQ(Int16 k, Int16 l, const IntLLR *parity, const IntLLR *extrinsic)


/*===============================================================================*/
/* UTILITY FUNCTION: JacLog                                                      */
/*-------------------------------------------------------------------------------*/
/* Description: Jacobian logarithm                                               */
/*                                                                               */
/* In:     IntLLR a  -> value one                                                */
/*         IntLLR b  -> value two                                                */
/* Return: IntLLR    <- Jacobian logarithm                                       */
/*-------------------------------------------------------------------------------*/
IntLLR JacLog(Int32 a, Int32 b)
```

4.4.4 PSAP transmitter functions

```
/*===============================================================================*/
/* PSAP FUNCTION: PsapTransmitter                                                */
/*-------------------------------------------------------------------------------*/
/* Description: PSAP downlink transmitter (uses prestored sequences)             */
/*                                                                               */
```

```
/* In:     Int16  symbol  -> one out of four downlink symbols                */
/*         Int16  index   -> position within downlink frame                  */
/* Out:    Int16* pcm     <- output data                                     */
/*-------------------------------------------------------------------------*/
void PsapTransmitter(Int16 *pcm, Int16 symbol, Int16 index)
```

4.4.5 IVS receiver functions

```
/*=========================================================================*/
/* IVS FUNCTION: IvsReceiver                                               */
/*-------------------------------------------------------------------------*/
/* Description: IVS receiver function                                      */
/*                                                                         */
/* In:     const Int16* pcm     -> input to downlink receiver              */
/* Out:    Int32*       metric  <- metric[k]: k = 0..3 correlation values   */
/*                              <- metric[4]: reliability factor (-1: skip) */
/* Return: Int16                <- demodulated message                     */
/*-------------------------------------------------------------------------*/
Int16 IvsReceiver(const Int16 *pcm, Int32 *metric)
```

4.4.6 Synchronization functions (IVS and PSAP)

```
/*=========================================================================*/
/* FUNCTION: EcallSync                                                     */
/*-------------------------------------------------------------------------*/
/* Description: main synchronization function                              */
/*                                                                         */
/* In:     const Int16* pcm      -> input frame                            */
/*         const char*  caller   -> text to identify PSAP or IVS           */
/* InOut:  SyncState*   sync    <-> sync struct                            */
/*-------------------------------------------------------------------------*/
void EcallSync(SyncState *sync, const Int16 *pcm, const char *caller)


/*=========================================================================*/
/* UTILITY FUNCTION: IvsRxSync                                             */
/*-------------------------------------------------------------------------*/
/* Description: IVS sync evaluation                                        */
/*                                                                         */
/* In:  const Int16* pcm  -> input frame of 16bit PCM samples              */
/*-------------------------------------------------------------------------*/
void IvsRxSync(const Int16 *pcm)


/*=========================================================================*/
/* IVS FUNCTION: EcallSyncCheck                                            */
/*-------------------------------------------------------------------------*/
/* Description: checks whether sync is still valid at IVS                  */
/*                                                                         */
/* In:     const Int16* pcm    -> input frame                              */
/* InOut:  SyncState*   sync  <-> sync struct                              */
/*-------------------------------------------------------------------------*/
void EcallSyncCheck(SyncState *sync, const Int16 *pcm)


/*=========================================================================*/
/* UTILITY FUNCTION: ToneDetect                                            */
/*-------------------------------------------------------------------------*/
/* Description: tone detection at 500 Hz or 800 Hz                         */
/*                                                                         */
/* In:     const Int16* pcm    -> input frame                              */
/* InOut:  SyncState*   sync  <-> sync struct                              */
/*-------------------------------------------------------------------------*/
void ToneDetect(SyncState *sync, const Int16 *pcm)
```

```
/*============================================================================*/
/* UTILITY FUNCTION: UpdatePeak                                               */
/*----------------------------------------------------------------------------*/
/* Description: update sync peak position                                     */
/*                                                                            */
/* In:     const Int32* pos   -> vector of positions                          */
/*         const Int32* corr  -> vector of correlation values                 */
/*         Int16        dist  -> distance to be checked                       */
/* Return: Int16              <- updated peak position                        */
/*----------------------------------------------------------------------------*/
Int16 UpdatePeak(const Int32 *pos, const Int32 *corr, Int16 dist)


/*============================================================================*/
/* UTILITY FUNCTION: CheckPosPeaks                                            */
/*----------------------------------------------------------------------------*/
/* Description: check positive sync peaks                                     */
/*                                                                            */
/* In:     const char*  caller        -> text to identify PSAP or IVS         */
/*         const Int32* pCorr         -> vector of correlation values         */
/*         Int16        p1            -> peak position p1                     */
/*         Int16        p2            -> peak position p2                     */
/*         Int16        ppPeaks       -> number correct pos/pos distances     */
/*         Int16        npPeaks       -> number correct neg/pos distances     */
/*         Int16        targetDelay   -> target delay if sync successful      */
/* InOut:  SyncState*   sync         <-> sync struct                          */
/*----------------------------------------------------------------------------*/
void CheckPosPeaks(SyncState *sync, const char *caller, const Int32 *pCorr,
                   Int16 p1, Int16 p2, Int16 ppPeaks, Int16 npPeaks,
                   Int16 targetDelay)


/*============================================================================*/
/* UTILITY FUNCTION: CheckNegPeaks                                            */
/*----------------------------------------------------------------------------*/
/* Description: check negative sync peaks                                     */
/*                                                                            */
/* In:     const char*  caller        -> text to identify PSAP or IVS         */
/*         const Int32* nCorr         -> vector of correlation values         */
/*         Int16        n1            -> peak position n1                     */
/*         Int16        n2            -> peak position n2                     */
/*         Int16        nnPeaks       -> number correct neg/neg distances     */
/*         Int16        npPeaks       -> number correct neg/pos distances     */
/*         Int16        targetDelay   -> target delay if sync successful      */
/* InOut:  SyncState*   sync         <-> sync struct                          */
/*----------------------------------------------------------------------------*/
void CheckNegPeaks(SyncState *sync, const char *caller, const Int32 *nCorr,
                   Int16 n1, Int16 n2, Int16 nnPeaks, Int16 npPeaks,
                   Int16 targetDelay)
```

4.4.7 Other utility functions (IVS and PSAP)

```
/*============================================================================*/
/* UTILITY FUNCTION: SetModState                                              */
/*----------------------------------------------------------------------------*/
/* Description: set the modulator state                                       */
/*                                                                            */
/* In:     Int16     modType   -> type of modulator to use                    */
/* InOut:  ModState* ms       <-> modulator struct                            */
/*----------------------------------------------------------------------------*/
void SetModState(ModState *ms, ModType modType)
```

Annex A (informative): Change history

Change history							
Date	**TSG SA#**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
2009-03	43	SP-090201			Approved at TSG SA#43	2.0.0	8.0.0
2009-06	44	SP-090251	0001	1	Correction of a mismatch with 3GPP TS 26.267 concerning synchronization	8.0.0	8.1.0
2009-06	44	SP-090251	0002	1	Correction concerning modulator initialization	8.0.0	8.1.0
2009-06	44	SP-090251	0003	1	Correction of a mismatch with 3GPP TS 26.267 concerning ACK transmission	8.0.0	8.1.0
2009-06	44	SP-090251	0004	1	Extension of eCall test setup to allow conformance testing of ACK messages	8.0.0	8.1.0
2009-06	44	SP-090251	0005	2	Separation of IVS and PSAP transmitter and receiver functions in the C-code	8.0.0	8.1.0

History

Document history		
V8.0.0	April 2009	Publication
V8.1.0	June 2009	Publication