digisolutions

DV-RPTR

USB / serial interface specification

for open source firmware version 1.10 to 1.69b

developer manual

22 January 2014

Table of contents

Scope

This document describes how the communication between a DV-RPTR board (with open source firmware installed) or similar and a PC or embedded computer works.

Physical connection

USB connection

The DV-RPTR uses the PID of Atmel **(0x03EB)** and the VID of the CDC example **(0x2307)**. It implements a 'communication device class' device and need no special handling like other USB serial devices. All 'physical' parameters (baudrate, stop-bits, flow-control) are ignored (don't care).

Serial connection

As 2nd possible interface a DV-RPTR has a real serial 2-wire connection (TxD, RxD; 5V tolerant CMOS level pins). This interface can be used, if no AMBE addon board is present.

The physical parameter are fixed to 115200baud 8N1, no flowcontrol.

Data- and packet Formats

All data send to or receive from a DV-RPTR are framed in packets. This packets are used to identify the beginning (time independent) and length of the payload data. Additional a 16bit checksum secures the rest on physical serial connections.

The packet structure is calles PCP2 (packet communication protocol 2). The previous version (PCP) is used on the modified Siemens C5 radios

PCP1

On Siemens C5, PCP is used for communication with the main µP via TTL-serial link (5V; 57600baud 8N1). The protocol is described in SC5BOS development manual.

The main features:

- split up into 10 ports (endpoints in C5)
- up to 256 byte payload
- CRC CCITT 16bits (ignore leading zeroes)
- short frames (3 byte length) with XOR check-byte

PCP2

The PCP2 has some improvements:

- optional split up into 8 endpoints (command-byte is now part of payload)
- up to 65535 byte payload possible
- only one kind of frame and one frame ID (byte **'0xD0')**
- CRC CCITT 16bits (ignore leading zeroes) same like PCP, check is optional for USB

PCP2 frame structure

Pos	Meaning
0	START-ID **0xD0**
1	Length lower byte (Length of payload only - offset 03 to first crc-byte)

2	Length upper byte (Length is in little endian - "PC ready")	
3	Payload first byte, should be used as CMD/Message/EP declaration	
4	Payload 2nd byte	
	...	
L1	CRC upper byte	(L1 is position Length +3)
L2	CRC lower byte	(L2 is position Length +4)

DV-RPTR communication

General

All messages are packed into a PCP2 frame and start with a command-byte, as defined below. The command-byte (**CMD**) is the first thing you must check of every message (after receiving and checking a PCP2 frame). So keep the logic in this way:

1. listen to frame-begin identifier
2. receive minimum 5 bytes
3. do a plausible check of frame-length (more than 2048 bytes are not plausible and can't handled by the DV-RPTR; less than 1 byte payload are not possible)
4. receive the complete frame
5. check CRC, if this was necessary (serial connection)
6. now we had a valid PCP2 frame received, now you can start to identify the message by reading the **CMD**.

Structure of command byte

Bits	Meaning
7	Response-Bit, its set to '1' in a reply of a "GET" cmd
6..4	Endpoint definition ("0" DFU bootloader, "1" D-Star Hotspot/Repeater, ...)
3..0	Message ID / Command / Cmd-Group: if 16 different cmds enough, use it as a ID, if not: preselector of a message-group, next byte is the ID (DFU-Bootloader and Hotspot uses a few cmds only)

Timing, gap and limit considerations

A PC program must be ensure that a PCP2 frame is transferred as a continuous block. The maximum allowed gap on a block transfer is **5ms**.

The maximum burst length of one or more packets should never exceed **2048** bytes total. Enabling the CRC function results in ignoring all defective packets with a wrong CRC.

Hotspot/Repeater Messages

Commands / Messages used

ID	Message name	Type	Description
0x10	RPTR_STATUS	**R/C**	Get status information or set mode -> answer
0x11	RPTR_GET_VERSION	**REQ**	Get version information -> answer
0x12	RPTR_GET_SERIAL	**REQ**	Get Serial Number -> answer (32bit number)
0x13	RPTR_GET_CONFIG	**REQ**	Get configuration blocks -> answer
0x14	RPTR_SET_CONFIG	**CMD**	Update configuration block -> answer (ACK/NAK)
0x15	RPTR_RXPREAMBLE	**MSG**	Preamble detected message
0x16	RPTR_START	**MSG**	A reception starts (START-PATTERN in bit-stream)
0x17	RPTR_HEADER	**MSG**	A header was received (header is decoded in payload)
0x18	RPTR_RXSYNC	**MSG**	A reception starts on a received frame sync pattern
0x19	RPTR_DATA	**MSG**	Voice data
0x1A	RPTR_EOT	**MSG**	A STOP-PATTERN, end-of-transmission received
0x1B	RPTR_RXLOST	**MSG**	Sync lost (no valid reception any more)
0x1C	reserved		for future use
0x1D	reserved		for future use
0x1F	RPTR_SFC	**CMD**	Do a special function call

Legend:
CMD Command to DV-RPTR, DV-RPTR answers with a ACK/NAK + optional additional data
REQ Request data from DV-RPTR, DV-RPTRv answers with the data wanted
R / C Request or Command, depends on additional data
MSG Message to or from DV-RPTR to transport D-Star data

Requests

Status (RPTR_STATUS)

Command-byte:	**0x10**
Parameters:	**none**

Use this request, to get information about the current status of the DV-RPTR modem.
Request message to DV-RPTR: **0x10**
As complete PCP2 packet: **0xD0 0x01 0x00 0x10 crch crcl**

Answer from DV-RPTR:
Every answer contains the command-byte with OR-ed bit 7 as replied message ID. The first of the received data is in this case **0x90** (0x10 OR 0x80).

Pos	Name	Bits	Description
0	Message ID	8	contains always 0x90
1	state flags	16	bitset – see below
3	TX state	8	enumeration of transceiver state
4	receive-buffer-size	8	number of voicedata buffer for HF reception (default: 21)
5	transmit-buffer-size	8	number of voicedata buffer for transmission (default 12*21)
6	unsend counter	8	number of not transmitted voicedata
7	actual TX buffer index	8	current position of the while transmitting *(append in FW 1.11)*

Flags (Bitset):
Bit0 : Receiver enabled
Bit1 : Transmitter enabled
Bit2 : PC Watchdog enabled
Bit3 : Checksum-Calculation enabled (check packets received from PC)
Bit4 : **obsolete** I/O 21 Status (w/o addon board used as Jumper-Configuration)
Bit5 : **obsolete** I/O 23 Status (w/o addon board used as Jumper-Configuration)
Bit6 : rsvd
Bit7 : (1) physical layer not configured (C0 config see below).
Bit8 : Receiving
Bit9 : Transmitting (PTT enabled)
Bit10: PC Watchdog occured! (RX/TX disabled)
Bit11: PCP2 Checksum checked on reception

TX state enumeration

state	meaning
0	disabled; Tranceiver not active
1	idle; DV-RPTR don't transmit
2	TX delay; PTT line active, no modulation
3	transmitting 010101... preamble (end with start pattern)
4	header transmission (in this state the app can't update the header with another new one)
5	voice data
6	last frame transmission (voice followed by a stop pattern instead of slow data)
7	EOT transmit end-of-transmission

In source:

```
typedef enum {
  RPTRTX_disabled, RPTRTX_idle, RPTRTX_txdelay, RPTRTX_preamble,
  RPTRTX_header, RPTRTX_voicedata, RPTRTX_lastframe, RPTRTX_eot
} trptr_tx_state;
```

Version (RPTR_GET_VERSION)

Command-byte:	**0x11**
Parameters:	**none**

This command returns the 16bit BCD coded firmware version as a 16bit little-endian word followed by a short device identification (ASCII string).

Hints

The reply can differ in length, because the length of the identification isn't fix. The device identification don't contain a length byte (Pascal short string) nor a Null termination byte (C style). The length can never exceed 252 bytes but for static buffer a value of 32-48 bytes is practically.

Reply-Example

91 92 16 "DV-RPTR R. 2013-12-13"

How we should the version **0x1692** interpreted and displayed?

Highest nibble: Main Version **1**
Third nibble: Subversion **6** (increased if the firmware became new features)
Second nibble: Subsubversion **9** (increased if parts of the source are rewritten or optimized)
Lowest nibble: Bugfixlevel **b** (increased with one or more bugfixes applied at the same time)

Display it like *"V1.50a"*. A zero in the last nibble means "no bugfix level" (no letter).

Serial number (RPTR_GET_SERIAL)

Command-byte:	**0x12**
Parameters:	**none**

Returns a unique serial number (32bit). This number is programmed with the bootloader into the proteced bootloader area. Use this value to identify DV-RPTER if you have more than one connected.

Hints

Not all devices contains a valid serial number... If you using the firmware w/o a bootloader (JTAG debugging) you read a Zero with this command. Handle Zero and **0xFFFFFFFF** not as valid numbers.

Information from a serial number

A serial number is BCD coded and contains the year (first byte), the calendar week (2nd byte) folloed by a 'D' nibble and a 3 digit BCD counter (up to 999 devices per week).

Configuration (RPTR_GET_CONFIG)

Command-byte:	**0x13**
Parameters:	none or config block ID

Reads out all configuration blocks used from DV-RPTR (hides blocks of unequipped add-ons).

Read a specified configuration block

using GET command:
Send it w/o a parameter, all implemented config blocks a transfered to the pc.
Send it with 1 byte parameter (config-block-id), only one block (or NAK if not available) transferred back.

Commands

Mode (RPTR_SET_STATUS)

Command-byte:	`0x10`
Parameters:	`1 control byte`

This important command us used for enable or disable various firmware modules. It's the same command byte like “RPTR_GET_STATUS” but now with one byte as parameter as the single difference. This Parameter shout contain a Bitset where a '1' mean “enabled” and '0' “disabled'”. There are 4 modules defined:

Bit0: Enable/Disable Receiver ('1': power up ADC and enable FW reception function)
Bit1: Enable/Disable Transmitter ('1': power up DAC circuits, enable FW logic for TX)
Bit2: Enable/Disable PC Watchdog ('1' w/o PC communication RX/TX will be disabled after timeout)
Bit3: Enable/Disable Checksum-Calculation (“1” all follow packets must contain a valid crc)

-> Command returns ACK if success or NAK.

A 'NAK" reply mean that something goes wrong on this DV-RPTR (hardware fault)... It's not implemented till V1.69b.

Configuration (RPTR_SET_CONFIG)

Command-byte:	`0x14`
Parameters:	list of (supported) config blocks

like GET_CONFIG, but to put the configuration into DV_RPTR.
You can combine all blocks together or just send one or some blocks.
-> this command returns a ACK, if configuration was accepted, otherwise NAK.

Special Function Call

The SFC's (special function calls) are a group of various requests and commands used for control additional hardware, testing the DV-RPTR or check the capabilities. All sub-commands are highly depends on the firmware version (newer firmware = more SFCs). The PC application **MUST** check the firmware version first to decide what SFC can be used.

Configuration (RPTR_SET_CONFIG)

Command-byte	**0x1F**
SFC byte	See list below
Parameter	Various, depends on SFC

like GET_CONFIG, but to put the configuration into DV_RPTR.

SFC list

SFC_TRX_CAPABILITIES

min. Version	**1.69**
SFC byte	**0xC1**
Parameter	none

Returns the 4 byte long bitset of the internal radio, if installed (zero means = no internal TRX).

SFC_DGL_CAPABILITIES

min. Version	**1.69**
SFC byte	**0xC2**
Parameter	none

Returns the 4 byte long bitset of the AMBE add-on (zero means = no AMBE add-on board found).

SFC_SWITCH_DATA_PORT

min. Version	**1.70**
SFC byte	**0x01**
Parameter	1 Byte

Parameter: 1 Byte, Bit 0 decides the active port for TX / RX messages (1 = serial).
Returns ACK or NAK message (with Cmd + SFC prefix).

SFC_GET_CURR_RSSI

min. Version	**1.69**
SFC byte	**0x08**
Parameter	none

Returns message containing the current RSSI value.

SFC_CORRECT_TX_CLOCK

min. Version	**1.69**
SFC byte	**0x10**
Parameter	1 long int (32bit)

Tune the soft PLL for generating the 4800baud transmission clock + / - 131071 ticks. Non permanent.
Returns ACK or NAK message (with Cmd + SFC prefix).

SFC_TWI_WRITE

min. Version	**1.69**
SFC byte	**0x21**
Parameter	TWI address + data

Returns ACK or NAK message (with Cmd + SFC prefix) asynchronously.

SFC_DGL_MANUALPTT

min. Version	**1.69**
SFC byte	**0x40**
Parameter	1 byte

Bit 0 of the parameter controls the PTT, set to '1' results in start a transmission like the user has pressed the PTT button on the connected microphone.

SFC_DGL_VOLUMECTRL

min. Version	**1.69**
SFC byte	**0x41**
Parameter	2 byte

With this SFC the PC application controls volume or gain w/o changing the configuration. The fist parameter is a selector:
0 = DAC (main) volume (effects both outputs)
1 = Headset volume (used to set relative volume, speaker volume unchanged)
2 = Speaker volume (headset volume unchanged)
3 = ADC microphone gain
The second parameter is a signed byte in 0.5dB resolutions. Keep in mind, that all three value has differed borders (see TLV320AIC3120 datasheet for details).

SFC_DGL_ADCFILTER

min. Version	**1.69**
SFC byte	**0x44**
Parameter	various

SFC_DGL_ERRREG

min. Version	**1.70**
SFC byte	**0x4F**
Parameter	none

Returns the 4 byte long firmware error counter (counts communication errors between µC and AMBE-2020 or TLV320AIC3120).

SFC_DEFAULT_CONFIGS

min. Version	**1.69**
SFC byte	**0x80**
Parameter	none

Load all configuration blocks from EEProm (only existing). Restores start-up configuration.

SFC_STORE_CONFIC

min. Version	**1.69**
SFC byte	**0x81**
Parameter	none

Write all configuration blocks into the EEProm (skipping block for not existing optional hardware).

SFC_RESET

min. Version	**1.69**
SFC byte	**0xF1**
Parameter	4 byte key (serial number)

Do a software reset of the DV-RPTR (only if serial number is correct).

SFC_FORCE_BOOTLOADER

min. Version	**1.69**
SFC byte	**0xF2**
Parameter	4 byte key (serial number)

Set the permanent “force bootloader” flashrom fuse and do a software reset.
Warning: After this SFC, the PC must update / flash a firmware intro the DV-RPTR to set the device to normal operation. Without this, the DV-RPTR jumps into the DFU bootloader every power-up!

Transmitting D-Star Data

A valid D-Star stream contains several 1-time transmitted fields and the repeated 20ms voice-data block. For details see JARL protocol specification. The structure on HF looks like this:

Preamble (>= 32bit)	Start-Pattern (15 bit)	Header with FEC (660bit)	AMBE-Data (72bit)	Slow-Data (24bit)

AMBE-Data (72bit)	Slow-Data (24bit)	...	AMBE-Data (72bit)	Stop-Pattern (32+15bit)

First of all the Transmitter must be **enabled** with the RPTR_SET_STATUS command. Sending messages for a transmission w/o enabling results in **NAK** answers to every message.

To start a new D-Star transmission you can use the **RPTR_START** or the **RPTR_HEADER** message.

Usage and benefit of the optional **RPTR_START** message:

In the situation of a repeater or a multi-HF (more than one DV-RPTR) configuration you can detect the beginning reception 138ms before you got the decoded header-data. Now it makes sense to start the transmitter to reduce the system-delay and mark the TX channel as “busy”. The DV-RPTR keeps up to 420ms after the TX-Delay transmitting the preamble. Within this time you should send the **RPTR_HEADER** message.

Once **RPTR_HEADER** is transmitted to the DV-RPTR the “indexed buffer” logic must be used to transfer **RPTR_DATA** messages.

If the stream ends, you send a **RPTR_EOT** message to the DV-RPTR.

Stream counting

Every DV stream you send to the modem contains a ID (one byte number). This stream ID isn't transferred over HF. It helps to separate the streams on the DV-RPTR modem. The DV-RPTR is capable to transmit only one stream at the same time. A new stream must contain a new stream ID different from the previous one. Beginning a stream with **RPTR_START** / **RPTR_HEADER** you select a new ID (for example up-counting).

After this message all **RPTR_DATA** messages with another ID will be ignored: If some process (playback...) keep transmitting data it doesn't matter. Only the valid **RPTR_DATA** will be processed.

Considerations

What happen, if the DV-RPTR already transmits and gets a new (new stream ID) **RPTR_START** message?

Answer: Nothing: This message will be ignored.

What happen, if the DV-RPTR already transmits and gets a new **RPTR_HEADER** message?

Answer: The DV-RPTR stops the current stream after finishing the current voice-data (within a span of 0-20ms), transmits the Stop-Pattern, a preamble followed by a new Start-Pattern and the new header. This behaviour is like a break.

How the break can prevented?

For every “normal” transmissions you should check the TX state enumeration (**RPTR_STATUS** request). Example: A emergency or break transmission should result in an immediately start of this stream. A local stream you can handle in this way too, but on normal UDP streams you should consider the TX state and begin with the new on if TX state is “last frame”, “eot” or “idle”.

PTT Logic

The PTT output is updated by the DV-RPTR firmware automatically. All necessary timings (TX delay, EOT tail) are done by the firmware. It's possible to deactivate the PTT permanently (for using the AMBE add-on for example).

Indexed Buffer Logic

One of the most difficult points is the realized logic to transfer voice data packets into the DV-RPTR. This logic eliminates the need of buffers and sorting voice packets on the PC application.

By default the DV-RPR offers the PC a transmit buffer of 252 (12 x 21) positions for 20ms voice-data. The PC application should read out the value using the RPTR_STATUS command. After a transmission is established (**RPTR_HEADER** message **0x18**) the PC application 'copies' the voice-data into this buffer using the index-byte. All buffer positions are initialized with a silence frame in the moment a **RPTR_HEADER** message is handled.

The DV-RPTR begins 137.5ms + TX delay with sending from buffer index **0x00** regardless if it contains new data. Transmitting keeps established till a **RPTR_EOT** message is received or no new voice-data appear in the time the hole buffer is transmitted (252 positions ^= 5040ms). With every new voice-data this timeout is updated.

A PC application that handles an UDP-RTP stream needs only a timestamp-to-index calculation. If you got data with 0-to-20 (frame-sync-length) index numbers needs to add a “block offset” updated if a 'proofed' new packet contains index lower than the last 'in line' packet.

Messages for Transmitting

RPTR_START (0x16)

An optional control-message. Only needed to shorten the delay in a repeater / duplex configuration. It can be generated on reception of a start-message of the same or another DV-RPTR.

Detailed format: See next chapter (*Receiving D-Star Data*).

RPTR_HEADER (0x17)

Transfers the header-data to the DV-RPTR and start sending (if idle). The transmit buffer is cleared with silence frames. In the case a previous transmission is active on HF, it will be break imminently on the current frame and append an EOT pattern + a synchronization gap before the start / header will be transmitted. Detailed format: See next chapter (*Receiving D-Star Data*). Control flags and biterror count are not used for transmission.

Hints: Take care, if the PC application needs to break a running transmission. The header data can't be updated if the DV-RPTR transmitting a previous header (TX state is RPTRTX_header)!

RPTR_DATA (0x19)

Transfer digital voice and slow data into the transmit buffer using the index number. See “Voice Data Message” for details.

RPTR_EOT (0x1A)

End-Of-Transmission. Stops transmission at a defined end position (0 to 251). Using the key value 0xFF as position (recommended) the DV-RPTR stops with transmission of the last unsent voice-data (slow-data of this buffer will **not** send out in this case).

PCP2 header (3 byte)	Message ID (1 byte)	Stream ID (1 byte)	Stop position (1 byte)	PCP2 Checksum (2 byte, BE)

Voice Data Message

PCP2 header (3 byte)	Message ID (1 byte)	Stream ID (1 byte)	Buffer Index (1 byte)	unused (1 byte)

AMBE Voice Data (9 byte)	Slow Data (3 byte)	unused (1 byte)	Reserved (1 byte)	PCP2 Checksum (2 byte, BE)

Multi Voice Data Messages

Multiple voice-data messages are useful for playback recorded messages. They looks like the single voice data message, but the AMBE voice data + slow data are repeated up to 252 times (size of the transmit buffer). The buffer index bytes points to the first buffer.

Transmission Example

1. Request Status of DV-RPTR, check the TX state to prevent clipping the previous transmission.
2. Send START, if you got a RPTR_START from another device (not used for UDP reception).
3. Send **HEADER**.

   The transmitter turns on, if off. If TX on a "BREAK" (EOT->Sync) was generated.
4. Transfer **VOICEDATA**.

USE the same stream ID you used on HEADER / START messages. Ignore gaps in incoming data, it's automatically filled with silence data.

5. Send **EOT**-Cmd with index 0xFF

This generates a STOP PATTERN just after the last VOICE-packet. After this the PTT goes off.

Receiving D-Star Data

First of all the receiver (demodulator algorithm) must be **enabled** with the RPTR_SET_STATUS command (not needed for interal AMBEaddon generated streams). Without enabling the DV-RPTR consumes a bit less power.

While the reception control bit is turned on (see **RPTR_SET_STATUS**), a DV-RPTR forward received D-Star data to the computer immediately.
The same PCP2 frame-types like transmit are used. They are a few more information fields and a different numbering of VoiceData frames in comparison to transmit.

All messages contains an up-counting stream ID byte as first data after the message ID. A valid D-Star reception needs 3 different packet-types:
1. control data messages (with different message ID's)
2. header-data message (a single, optional message)
3. voice-data message (repeated every 20ms)

Why using a stream ID byte?

In some configurations it's possible to get two streams in parallel from the DV-RPTR. So you can configure a AMBE Addon board in “talk to internet” mode, keeping the duplex feature an. Now you can “talk to the internet” in the same time the DV-RPTR receives another stream from a radio (HF). An application should consider this, like a multi DV-RPTR environment.

Control-Data message

PCP2 Frameheader (3 byte)	Message ID (1 byte)	Stream ID (1 byte)	Reserved (1 byte)	PCP2 Checksum (2 byte, BE)

The DV-RPTR can create 5 different Control-Data messages. They appear only on beginning or at the end of a D-Star reception; never between header or voice-data messages.
In source:

```
typedef struct PACKED_DATA {
  tRS232pktHeader head;
  unsigned char   rxid;
  unsigned char   rsvd;
  unsigned short  crc;
} tctrlpcdata;
```

On beginning:

RPTR_RXPREAMBLE (0x15)

A preamble (alternating 1010...) is detected. That means 'channel busy'. After this message a RPTR_START follows (or nothing if reception are interrupted). The containing stream-ID is not valid here.

RPTR_START (0x16)

The uniqe 15 bit start pattern was received, after 137.5ms (+ approx. 1ms delay) a RPTR_HEADER message will follow. Take the stream ID from this message.

RPTR_RXSYNC (0x18)

The reception catch up an already running D-Star transmission (no header received) with receiving the 24 bit long frame-sync pattern. After 20ms a voice-data message follows.

The end of a transmission is noticed by one of the followed messages:

RPTR_EOT (0x1A)

End-Of-Transmission. Reception of the 15-bit stop pattern. The reserved byte contains the number oft the previous voice-data counter.

RPTR_RXLOST (0x1B)

Lost a transmission. The HF signal is to weak or no frame-sync patterns can be found in the bit-stream. The reserved byte contains the number oft the previous voice-data counter.

Between this messages you can get:

RPTR_HEADER (0x17)

Contains the deinterleaved and decoded D-Star header plus a few additional infos.

RPTR_DATA (0x19)

Every 20ms a voice-data frame is transmitted to the PC containing a frame-cointer, the unaltered HF data plus a few informations).

Header Data Message

PCP2 header (3 byte)	Message ID (1 byte)	Stream ID (1 byte)	Control-Flags (1 byte)	Biterror Count (1 byte)	Source Flags (1 byte)

D-Star header (Flags, RPT2, RPT1, YOUR, MY, MY2 CRC; 41 byte)	Reserved (1 byte)	PCP2 Checksum (2 byte, BE)

In source:

```
typedef struct PACKED_DATA {
  tRS232pktHeader head;       // 4 byte (ID, length and message-type)
  unsigned char   rxid;       // ID of current transmission
  unsigned char   flags;      // control-flags
  unsigned char   biterrs;    // no of bit-errors in a received header (not jet)
  unsigned char   srcflags;   // source flags, bit0 = internal by AMBEaddon
  tds_header      header;     // header starts at offset +8! 41 bytes long
  unsigned char   rsvd2;      // for future use
  unsigned short  crc;
} theadrpcdata;

typedef struct PACKED_DATA {
  unsigned char flags[3];
  char      RPT2Call[8];
  char      RPT1Call[8];
  char      YourCall[8];
  char      MyCall[8];
  char      MyCall2[4];
  unsigned short CRC;
} tds_header;
```

Control-Flags

Bit 7: Header corrupted.

Bits 6-0: not used.

Check Bit 7 every header message to find out if the header is valid. In the case of “1” the header contains garbage (FEC failed) and can't used. Often the follow voice data is garbage too.

Bit Errors

Number of corrected bit-errors. This test need an additional check and is not implemented <= V1.69 jet. In newer firmware (containing a stand-alone repeater mode) this feature is switch-able by an SFC.

Source Flags

Bit 7-1: not used (for future use)

Bit 0: Shows “1” if this stream was internally generated (by AMBEaddon).

Voice Data Message

PCP2 header (3 byte)	Message ID (1 byte)	Stream ID (1 byte)	Packet Counter (1 byte)	RSSI / SQL (1 byte)

AMBE Voice Data (9 byte)	Slow Data (3 byte)	Source Flags (1 byte)	Reserved (1 byte)	PCP2 Checksum (2 byte, BE)

In source:

```
typedef struct PACKED_DATA {
  tRS232pktHeader head;       // 4 byte (ID, length and message-type)
  unsigned char   rxid;       // ID of current transmission
  unsigned char   pktcount;
  unsigned short  rssi;       // SQL-line value (averaged for every voice frame)
  tds_voicedata   DVdata;     // DVdata starts at offset +8!
  unsigned char   srcflags;   // source flags, bit0 = internal by AMBEaddon
  unsigned char   rsvd;       // for future use
  unsigned short  crc;
} tvoicepcdata;

typedef struct PACKED_DATA {
  union {
    unsigned int packet[3];
    struct {
      unsigned char voice[9];
      unsigned char data[3];
    };
  };
} tds_voicedata;
```

Packet Counter

In difference to a stream sending to from PC to the DV-RPTR, this counter counts synchronous with the D-STAR frame-sync pattern (0 to 20). The value 'zero' contains (or should contain) the FRAME-SYNC within the 3 slowdata bytes ('value' of error-less frame-sync: **55 2D 16**). The application can read out the number of packets for a complete 'sync-run' with the **RPTR_STATUS** request (= Receive-Buffer-Size).

RSSI / SQL line value

A 2nd AD channel of the DV-RPTR samples a 0 to 5V signal connected to the SQL pin of the MiniDIN6 socket. Some receivers puts a digital squelch signal to the data socket. Some radio modules have an RSSI output. In both cases an analog (16bit LE) signal is digitized and averaged for every frame. A PC software can evaluate this for an additional squelch or visualize it.

This field contains 0x0000 in the case of an internal stream (AMBEaddon) or if unconnected (= 0V).

Source Flags

Same as header data message: Bit 0 = internal stream.

Reception Example

A typical reception look like this:

```
D0 | 03 00 | 15 | 00 00 crc       >Preamble detected (only, if long enough)
D0 | 03 00 | 16 | 01 00 crc       >Start-Frame-Pattern detected
D0 | 2C 00 | 17 | 01 00 00 00 {header data} crc
D0 | 0F 00 | 19 | 01 00 23 01 {VoiceData} 55 2D 16  crc >voicedata + framesync
D0 | 0F 00 | 19 | 01 01 25 01 {VoiceData}{SlowData} crc >slowdata not
descrambled
D0 | 0F 00 | 19 | 01 02 FE 00 {VoiceData}{SlowData} crc
D0 | 0F 00 | 19 | 01 03 0A 01 {VoiceData}{SlowData} crc
D0 | 0F 00 | 19 | 01 04 0C 01 {VoiceData}{SlowData} crc
...
D0 | 0F 00 | 19 | 01 00 2A 01 {VoiceData} 55 2D 16  crc
D0 | 0F 00 | 19 | 01 01 21 01 {VoiceData}{SlowData} crc
                          ^ RSSI value
D0 | 03 00 | 1A | 01 01 crc        >End of Transmission
                  ^ stream ID 0..255
               ^ message ID
      ^ PCP2: Length of Data (packet length - 5)
^ PCP2: FrameID
```

DV-RPTR configuration

Structure of a configuration block

To have the option for expanding the configuration in later firmware versions, I defined a very simple block format to separate config-values in logical units. Every configuration block starts with an uniqe ID byte (from 0xC0 to 0xCF) followed by the length of the configuration (1 byte):

<config-block-id> <blocksize> <config-block>

All data value greater 1 byte are expected in little endian (PC) format and not in big-endian (the AVR32 controller uses big endian internally).

List of configuration blocks

ID	Length	Configuration
0xC0	4	Basic DV-RPTR configuration (physical radio parameters)
0xC1	12	Internal or attached transceiver setup
0xC2	40	Dongle or stand alone repeater: header configuration and data
0xC3	20	Dongle or stand alone repeater: message text
0xC4	8	AMBE-Addon TLV320AIC codec basic configuration
0xC5	12	AMBE-Addon TLV320AIC codec AGC and DRC register configuration
0xC6	56	AMBE-Addon TLV320AIC codec custom filter block coefficients

Configuration details

Basic configuration (0xC0)

The Basic configuration contain the basic radio parameters.

Stucture (with config-block header)

0 **0xC0** Configuration ID
1 **0x04** Length of configuration data
2 **Flags** Bitset with the following meaning:

7	Halfduplex mode (1 = disables receiver while transmitting)
6	Dongle mode (1 = disabled PTT output pin while transmitting)
5	reserved
4	reserved
3	Automatic RX-Inversion detect (1 = enabled)
2	TX-Channel select: 0 = FSK pin (default), 1 = AFSK pin (channel B)
1	TX-Inversion
0	RX-Inversion

3 **Modulation** Voltage Peak-Peak (value 255 ^= 3.00V_{pp})
4..5 **TX-Delay** Delay between transmitter enable and start of transmitting preamble in milliseconds

Structure in source code

```
// configuration routines - physical config (MAIN config C0)
typedef struct PACKED_DATA {
```



```
  unsigned char  flags;
  unsigned char  mod_level;
  unsigned short txdelay;     // attention: little-endian here!
} t_config_0;
```

Transceiver setup (0xC1)

Till today only a RDA1846 based concept board exists and con be used as small transceiver. The only necessary values needed are the transmit- and receive-frequency. The presence of a connected transceiver can be evaluated using the SFC **`SFC_TRX_CAPABILITIES (0xC1)`**.

Stucture (with config-block header)

0	**0xC1**	Configuration ID
1	**0x0C**	Length of configuration data
3..6	**RX-Freq**	Receive frequency given in Herz [Hz]
7..10	**TX-Freq**	Transmit frequency [Hz]
11	**Flags**	Bitset with the following meaning:

7	
6	
5	
4	
3	
2	
1	
0	

12..14	reserved	For future use

Structure in source code

```
// configuration routines - physical config
typedef struct PACKED_DATA {
  unsigned int  rx_freq;      // in Hz little endian here
  unsigned int  tx_freq;      // in Hz little endian here
  unsigned char flags;        // bandwidth...
  unsigned char rsvrd[3];
} t_config_1;
```

Dongle header configuration (0xC2)

This configuration is only used at a stand alone configuration and for Microphone-triggered transmissions. In standard modem operation no header data needed to configure (a D-Star header is a part of the transmission data).
The presence or absence of the AMBE add-on the application can find out using the SFC **`SFC_DGL_CAPABILITIES (0xC2)`**.

0	**0xC2**	Configuration ID
1	**0x28**	Length of D-Star header data
2	**RPTR flags**	Dongle / Repeater control flags

7..3	reserved
4	AMBE NF microphone PTT switch enabled (not listen only)
3	AMBE NF receive roger beep
2	AMBE NF own PTT can break

1	AMBE NF listen to the internet (voice data from PC is audible)
0	AMBE NF listen to the radio (received voice data is audible)

3	**0x00**	Header Flag 1 (not used / internal setup)
4	**0x00**	Header Flag 2 (not used / internal setup)
5	**0x00**	Header Flag 3 (not used / internal setup)
6..13	**RPT2Call**	Header Repeater 2 value (8 chars)
14..21	**RPT1Call**	Header Repeater 1 value (8 chars)
22..29	**YourCall**	Header YourCall value (8 chars)
30..37	**MyCall**	Header own Call value (8 chars)
38..41	**MyCall 2**	Header own Call suffix (4 chars only)

Dongle message text (0xC3)

0	**0xC3**	Configuration ID
1	**0x14**	Length of message text (is fixed to 20 chars)
3..21	**message**	Message text

TLV320AIC basic configuration (0xC4)

The presence or absence of the AMBE add-on the application can find out using the SFC **SFC_DGL_CAPABILITIES (0xC2)**.

Stucture (with config-block header)

0	**0xC4**	Configuration ID
1	**0x08**	Length of configuration data
2	**mic_pga**	Microphone PGA Gain 0..59.5dB (0.5dB steps)
3	**mic_imp**	Microphone Impeadance and BIAS selection
4	**adc_gain**	ADC Gain value (digital) -12..20dB (0.5dB steps)
5	**adc_filter**	ADC filter selection PB4, PB5 or PB6
6	**spkr_out**	internal speaker -128 = off; gain 0.5dB steps
7	**hs_out**	handset speaker -128 = off; gain 0.5dB steps
8	**volume**	DAC volume (-63.5dB .. +24dB; 0.5dB steps) 0x7F = controlled by Pin
9	**dac_filter**	DAC filter selection

Microphone Impedance and BIAS

Bits		**Values**	
0..1	Bias value	**0**	No BIAS voltage (dynamic microphone)
		1	2V
		2	2.5V (default)
		3	3.3V
2..4	P-Terminal Impedance	**1**	Left only 40kOhm
		2	Left only 20kOhm
		3	Left only 10kOhm
		4	Left + Right 40kOhm

		5	Left + Right 20kOhm
		6	Left + Right 10kOhm
5..7	M-Terminal Impedance	**0**	40kOhm
		1	20kOhm
		2	10kOhm

In source:

```
typedef struct PACKED_DATA {
  unsigned char   mic_pga;    // Microphone PGA Gain 0..59.5dB (0.5dB steps)
  unsigned char   mic_imp;    // Microphone Impedance Value
  signed   char   adc_gain;   // ADC Gain Value -12..20dB (0.5dB steps)
  unsigned char adc_filter;   // Selection between PB4,PB5 and PB6
  signed   char   spkr_out;   // internal speaker -128 = off; gain 0.5dB steps
  signed   char hs_out;       // handset speaker -128 = off; gain 0.5dB steps
  signed   char volume;       // DAC volume (-63.5dB .. +24dB; 0.5dB steps)
  unsigned char dac_filter;   // DAC filter selection
} t_config_4;
```

TLV320AIC AGC and DRC configuration (0xC5)

In source:

```
typedef struct PACKED_DATA {
  unsigned char agc_ctrl1;    // Automatic Gain Control (Bit7 enable)
  unsigned char agc_ctrl2;    // see TLV320AIC datasheet (register pge0,0x56ff)
  unsigned char agc_maxgain;  // maximum gain 0..59.5dB (0.5dB steps)
  unsigned char agc_attack;   // attack time
  unsigned char agc_decay;    // decay time
  unsigned char agc_noisedeb; // noise debounce
  unsigned char agc_signaldeb; // signal debounce
  unsigned char rsvd1;
  unsigned char drc_ctrl1;    // Dynamic Range Compression (Bit6 enable)
  unsigned char drc_ctrl2;    // see TLV320AIC datasheet (register
pge0,0x44,0x45,0x46)
  unsigned char drc_ctrl3;
  unsigned char rsvd2;
} t_config_5;
```

TLV320AIC filter coefficients config (0xC6)

In source:

```
typedef struct PACKED_DATA {
  S16 FirstOrderIIR[3];
  S16 BQBlockA[5];
  S16 BQBlockB[5];
  S16 BQBlockC[5];
  S16 BQBlockD[5];
  S16 BQBlockE[5];
} tfilter_config;
```

