

AMMOS Datastreams

Reference Document



4094.8964.02 – 04



ROHDE & SCHWARZ

Radiomonitoring & Radiolocation

Reference Document

© 2014 Rohde & Schwarz GmbH & Co. KG

Mühlhofstr. 15, 81671 München, Germany

Phone: +49 89 41 29 - 0

Fax: +49 89 41 29 12 164

E-mail: info@rohde-schwarz.com

Internet: www.rohde-schwarz.com

Subject to change – Data without tolerance limits is not binding.

R&S® is a registered trademark of Rohde & Schwarz GmbH & Co. KG.

Trade names are trademarks of the owners.

Contents

1	Generic Format of Datastreams	5
2	IF (Baseband) Datastreams	8
2.1	IF Data Format	8
2.2	IF DDCE Data Format	16
3	Audio Datastream	20
4	Tuner Datastreams	24
4.1	Scan Data Format	24
4.2	Signal Level Indicator Data Format	27
4.3	Tuner PIF Panorama Data Format	28
4.4	Tuner HF (EM010) Data Formats	28
5	Spectrum Datastreams	34
5.1	Spectrum Data Format	34
5.2	Segmentation Spectrum Data Format	37
6	Symbol Datastreams	39
7	Time Domain Datastreams	42
7.1	Time Domain Data Format	42
7.2	Instantaneous Data Format	43
8	Decoder Datastreams	46
8.1	Image Data Format	46
8.2	Decoded Text Data Format	49
8.3	Transmission System Result (TSR) Data Formats	51
9	Detector Datastreams	70
9.1	Emission List Data Format	70
9.2	Spectral Detector List Data Format	72
9.3	Burst Emission List Data Format	74
10	Statistics Datastreams	77
10.1	Histogram Data Format	77
10.2	Hop Density Waterfall Data Format	79

11	PDW and IQDW Datastreams.....	82
11.1	Pulse Descriptor Words (PDW) Datastream.....	82
11.2	IQ Descriptor Words (IQDW) Datastream.....	84
A	Extras.....	88
A.1	Data types definitions.....	88
A.2	File Types.....	89
	Glossary.....	92
	Index.....	94

1 Generic Format of Datastreams

The document describes digital transmissions between the following Rohde & Schwarz® radio monitoring and radiolocation devices for signal analysis: R&S CA100, R&S CA120, R&S GX400, R&S GX410, R&S GX430, R&S GX420, R&S GX425, R&S GX460, R&S GX465, R&S CA250, R&S TPA.

All datastreams described in this document have the same generic format. They are structured into frames having a generic frame header and, depending on data type to be transmitted, specific payload elements.

Generic Frame format

All datastreams have a frame based structure using the same format, consisting of a global *Frame header* coupled with a data-type specific *Frame body* (i.e. the frame payload).

The header and the body of the frame consist of a number of 32-bit words. The *Frame header* has a predefined structure and size. The size and structure of the *Frame body* depends on the payload type. This is an important factor in the choice of the frame size.

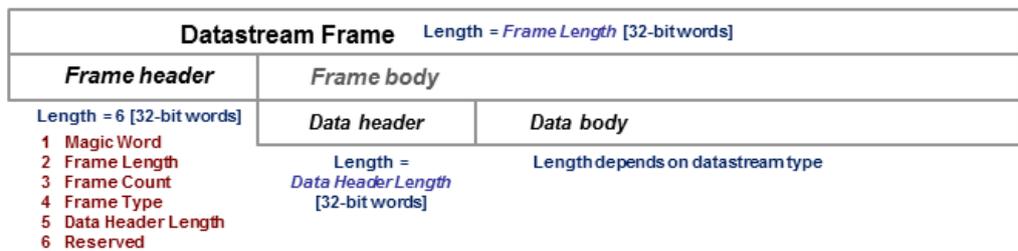


Fig. 1-1: Generic Datastream Frame structure

Global Frame header

The *Frame header* contains information used for frame synchronization, frame sequencing, payload identification and frame sizing. It consist of six 32-bit words as depicted in the following figure and is defined in `rs_gx40x_global_frame_header_if_defs.h`

Table 1-1: Global Frame header (structure name: `typFRH_FRAMEHEADER`)

Word position in frame	Member name Member type	Description
1	uintMagicWord ptypUINT	Magic Word - 32-bit word, always identical (0xFB746572), defines the start of the <i>Frame header</i> and is used for frame synchronization. The <i>Magic Word</i> and the <i>Frame Length</i> are used to identify the beginning of each frame.
2	uintFrameLength ptypUINT	Frame Length - gives the length of the frame including both <i>Frame header</i> and <i>Frame body</i> . The length is expressed in 32-bit words. The minimum length is six in case the <i>Frame body</i> is empty and the maximum length is limited to the value: <ul style="list-style-type: none"> • <code>kFRH_FRAME_LENGTH_MAX = 0x100000</code> ($1048576 = 2^{20}$) in case of normal frames • <code>kFRH_FRAME_LENGTH_MAX_EX = 0x400000</code> ($64 * 1048576 = 2^{26}$) in case of extended frames (an extended frame is marked by Bit#0 of the Reserved word of the frame header). Only some datastream types allow the extended frame size, see the definitions in the <code>rs_gx40x_global_frame_types_if_defs.h</code>. The next <i>Magic Word</i> which denotes the next frame in this data stream will occur <code>uintFrameLength</code> [32-bit words] after the <i>Magic Word</i> in this frame.
3	uintFrameCount ptypUINT	Frame Count - sequence counter modulo 2^{32} . Determines the position of this frame in the datastream and is used for sequencing and lost frame detection.
4	uintFrameType ptypUINT	Frame Type - identifies the data type contained in this frame and gives the specific structure of the frame payload. The complete list of frame types (i.e. datastream types) can be found in the following header file: <code>rs_gx40x_global_frame_types_if_defs.h</code>
5	uintDataHeaderLength ptypUINT	Data Header Length - gives the length of the <i>Data header</i> positioned at the beginning of the <i>Frame body</i> . The length is expressed in 32-bit words (0 means no data header). This information can be used by the software to recognize the version of the datastream format and thus its compatibility to read and correctly interpret the datastream. It enables forward-compatibility with future datastream versions. This value will not vary for a continuous data stream.
6	uintReserved ptypUINT	<ul style="list-style-type: none"> • Bits #31 to #1 - Reserved (not yet used, must be 0) • Bit #0 - Marks the frame with extended size (up to <code>kFRH_FRAME_LENGTH_MAX_EX</code> 32-bit words).



The *Data Header Length* information is very important for the correct addressing of the data samples. This information gives the exact position in the frame where the *Data body* begins independent of the version of the *Data header* (different versions consist of different number of parameters). From the frame beginning (indicated by the *Magic Word*), the first six 32-bit words represent the *Frame header* and the next *Data Header Length* 32-bit words represent the *Data header*. After $6 + \text{uintDataHeaderLength}$ 32-bit words starts the *Data body*, i.e. the first data sample.

Frame body

The *Frame body* contains the payload of the frame and its structure depends on the datastream type, as defined by the *Frame Type* element in the *Frame header*.

The *Frame body* is structured into a *Data header* followed by the *Data body*. The *Data header* contains datastream specific information of the payload.

NOTICE**Bit numbering**

Allover this document it is assumed that bit #0 is the bit of least numeric significance.

NOTICE**Data types definitions**

The datastreams described in this document use custom data type definitions, such as `ptypUINT`, that ensure the 32-bit format on all platforms. Detailed information about the custom data types used can be found in [chapter A.1, "Data types definitions"](#), on page 88.

2 IF (Baseband) Datastreams

2.1 IF Data Format

Intermediate Frequency (IF) Data format is used for the transmission of real or complex baseband signals. The IF signal is sent along with information that characterize the datastream and datastream source, and contains also the inband signaling of the signal processing unit.



As most of the received signals are I/Q (complex) baseband signals, this documentation uses the terms IF and I/Q interchangeably.

The IF Data format is valid for the following datastream types:

Table 2-1: IF Datastream types

Datastream type ID	Description	Sample data type
ekFRH_DATASTREAM__IFDATA_32RE_32IM_FIX ekFRH_DATASTREAM__IFDATA_32RE_32IM_FIX_RESCALED	Complex IF Data samples, 32-bit real-part and 32-bit imaginary-part, fixed point	typIFD_SAMPLE_32RE_32IM_FIX
ekFRH_DATASTREAM__IFDATA_16RE_16IM_FIX	Complex IF Data samples, 16-bit real-part and 16-bit imaginary-part, fixed point	typIFD_SAMPLE_16RE_16IM_FIX
ekFRH_DATASTREAM__IFDATA_16RE_16RE_FIX	Real IF Data samples, 16-bit real-part, two samples in each 32-bit word, fixed point	typIFD_SAMPLE_16RE_16RE_FIX
ekFRH_DATASTREAM__IFDATA_32RE_32IM_FLOAT_RESCALED	Complex IF Data samples, 32-bit real-part and 32-bit imaginary-part, floating point	typIFD_SAMPLE_32RE_32IM_FLOAT

For the above datastream types, the same frame body structure is used, the only difference is the carried sample data type (as given in the table above).

IF Data Frame Structure

The structure of the IF Datastream is defined in the `rs_gx40x_global_ifdata_header_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

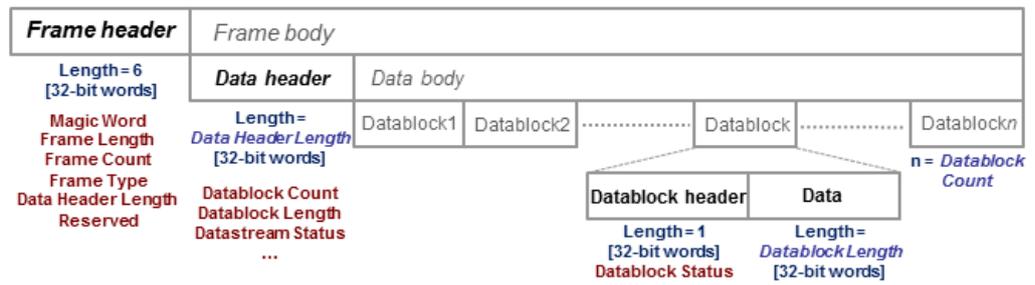


Fig. 2-1: IF Data frame format

NOTICE

Limited use of the frame structure definition: typIFD_IFDATA_FRAME

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

IF Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common properties of the Data samples.

The **basic** *Data header* contains a number of fields, that are always sent.

The **extended** *Data header* contains extra information fields sent after the fields of the basic structure.

The length of the *Data header*, as specified by the `uintDataHeaderLength` parameter from the *Frame header*, gives information about which *Data header* type is used, the basic or the extended one.

The IF *Data header* structure, of type `typIFD_IFDATAHEADER`, is described in the following table (Data header length = 14 [32-bit words]).

Table 2-2: IF DATA header (typIFD_IFDATAHEADER)

Word position in frame	Member name Member type	Description
7	uintDatablockCount ptypUINT	Datablock Count - represents the number of IF signal data blocks in the IF Data frame.
8	uintDatablockLength ptypUINT	Datablock Length - The number 32-bit words in each IF signal data block excluding the data block header (has to be of the form 2^N with $N \geq 2$). This may not be the same as the number of IF signal data samples, as the size of a sample may be 16, 32 or 64 bits.
9 10	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [μ s] - Absolute time of the first IF signal data sample, in the first data block of IF signal data in this frame.
11	uintStatusword ptypUINT	Status Word - extra information that help the receiver react by parameter changes. <ul style="list-style-type: none"> • Bit #31 - Reserved • Bit #30 - dBFS flag <ul style="list-style-type: none"> - 1 indicates that all samples in this frame are considered to be dBFS (dB full scale). - 0 indicates that the values <i>Antenna Voltage Reference</i> and <i>Reciprocal gain correction</i> (see the <i>Status Word</i> description of the datablock header) can be used to calculate the corresponding level for each sample. • Bits #29 to #8 - Reserved (not yet used, must be 0) • Bits #7 to #0 - User flags for special signaling between IF Data processing components.
12	uintSignalSourceID ptypUINT	Signal Source Identifier or antenna identifier (value 0x0 if not used)

Word position in frame	Member name Member type	Description
13	uintSignalSourceState ptypUINT	<p>Current Signal Source State (value 0×0 if not used)</p> <ul style="list-style-type: none"> gives the <i>Configuration Set Identifier</i> of the Task Data Set (in GX400) currently being applied by the IF signal source OR the current <i>Scan Step Number</i> in the case of scan operation In the case of memory scanning, the scan step number starts at 0 for the scan channel (memory location) configured with the lowest frequency, and increments (+1) for every channel configured for scanning in the memory scan list. In the case of frequency scanning, the scan step number starts at 0 for the scan step at the lowest frequency, and increments (+1) for every step taken within the configured frequency scan range.
14 15	uintTunerFrequency_Low uintTunerFrequency_High ptypUINT	64-bit Tuner Center Frequency [Hz] - least significant 32 bits (uintTunerFrequency_Low) followed by most significant 32 bits (uintTunerFrequency_High)
16	uintBandwidth ptypUINT	IF signal 3dB Bandwidth [Hz]
17	uintSamplerate ptypUINT	Sample Rate of the AD Converter [samples / second] - due to digital filtering within the source, the resulting sample rate of the samples within this frame is: Sample Rate × Interpolation / Decimation
18	uintInterpolation ptypUINT	Interpolation Factor referred to the ADC signal sample rate. The value 0×1 indicates no interpolation

Word position in frame	Member name Member type	Description
19	uintDecimation ptypUINT	Decimation Factor referred to the ADC signal sample rate. The value 0x1 indicates no decimation
20	intAntennaVoltageRef ptypINT	Antenna Voltage Reference (Ant-VoltRef) is the device specific correction value for the tuner front-end Rx attenuation (expresses anything from antenna input connector to ADC) and is expressed in [0.1 dB μ V]. This is the level which, while the AGC amplification is at maximum attenuation, is required at the antenna input to produce the full scale value at the ADC. Using this value together with the Recip-Gain (Reciprocal Gain) value, one can calculate the true signal level at the antenna input connector (see "Data samples" on page 14). The RecipGain value is given in the Status Word of the IF Datablock header table 2-4

The Extended IF *Data header* structure, of type `typIFD_IFDATAHEADER_EX`, is described in the following table (total Data header length = 19 [32-bit words]).

Table 2-3: Extended IF DATA header (typIFD_IFDATAHEADER_EX) - extra fields only

Word position in frame	Member name Member type	Description
21 22	bigtimeStartTimeStamp ptypBIGTIME_NS	64-bit Timestamp [ns] - Absolute time of the first sample of the datastream since starting the datastream ("Sample Counter" == 0). This value remains constant until the datastream is stopped and started again or until the tuner performs an internal synchronization.
23 24	uintSampleCounter_Low uintSampleCounter_High ptypUINT	Sample Count - 64-bit counter from the first sample of the first datablock in this frame. Note that this value can be reset when the datastream is stopped and started again or when the tuner performs an internal synchronization. The Sample Count of the next IF frame can be deduced from Datablock Count, Datablock Length and the number of 32-bit words per sample. In this way the number of sample Dropouts can be estimated (that can be replaced with Null values). The exact time is given by $t = Start\ Time + Sample\ Count * Decimation / (Sample\ rate * Interpolation)$.
25	intKFactor ptypINT	kFactor - Correction factor of the current antenna, given in 0.1dB/m. Used to determine the field strength (in dB μ V/m) at the antenna from the voltage level at the antenna input of the receiver. Contains antenna gain, cable attenuation, antenna switch matrix attenuation and anything else from air to antenna input. (the value 0x80000000 is used if no kFactor is defined).



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

IF Data Body

The IF *Data body* contains zero or more IF Data samples arranged as an array of `typIFD_DATABLOCK` data blocks (the actual IF signal datastream payload). The number of datablocks is specified by the Datablock Count parameter from the *Data header*.

Each datablock (`typIFD_DATABLOCK`) has its own datablock header: `datablockheaderDatablockHeader` (of type `typIFD_DATABLOCKHEADER`) and a datablock body that contains the actual data sample.

Table 2-4: IF Datablock header (`typIFD_DATABLOCKHEADER`)

Member name Member type	Description
<code>uintStatusword</code> <code>ptyUINT</code>	Status of the Datablock <ul style="list-style-type: none"> • Bits #31 to #16 - RecipGain - Automatic Gain Control (AGC) Reciprocal Gain Correction value that was applied when generating the following IF Data samples. The RecipGain is represented as 16-bit unsigned decimal value (the 16-bit unsigned decimal has to be divided by $2^{16} = 65535$ to obtain the unsigned fractional between 0 and 1). For example a correction value of -17.5dB gives a value for RecipGain of 0.1333 which will be represented as <code>0x2220</code>. Using this value together with the value for the antenna voltage reference, one can calculate the true signal level at the antenna input connector (see "Data samples" on page 14). • Bits #15 to #8 - Reserved (must be 0). • Bits #7 to #2 - User flags for special signaling between IF Data processing components. Set to 0 if not used. • Bit #1 - Blanking flag - this flag is set (1) to indicate that the data in this block may have been falsified by some external event. • Bit #0 - Invalidity flag - this flag is set (1) to indicate that the data within this block may be corrupt (e.g. the input signal exceeded the range of the AD converter, or the analog signal input from which the data was converted was overloaded), OR any one of the fields in the IF datastream header does not represent the data in this block correctly.

The datablock body is defined as an array of size `uintDatablockLength` with `uintData` elements interpreted using the corresponding sample type format ("`typIFD_SAMPLE....`" as described in the following table). The actual IF Data samples have to be extracted from the array. Their structure and size is given by the IF datastream format ([table 2-1](#)). The possible IF data sample formats are described in the table below:

Table 2-5: IF Data sample format

Sample type	Sample format	Most significant bits	Least significant bits	Data type
<code>typIFD_SAMPLE_32RE_32IM_FIX</code>	64-bit I/Q format	First 32-bit Real component		<code>ptyINT</code> or <code>pty-FLOAT_SP</code>
<code>typIFD_SAMPLE_32RE_32IM_FLOAT</code>		Second 32-bit Imaginary component		<code>ptyINT</code> or <code>pty-FLOAT_SP</code>

Sample type	Sample format	Most significant bits	Least significant bits	Data type
typIFD_SAMPLE_16RE_16IM_FIX	32-bit I/Q format	16-bit Imaginary component	16-bit Real component	ptypINT
typIFD_SAMPLE_16RE_16RE_FIX	16-bit Real format	16-bit sample number I+1	16-bit sample number I	ptypINT

The term 'fix' ('fixed' point) indicates signed (2s-complement) fixed point fractional numbers.

Data samples

The absolute signal level in [dBμV] may be calculated as follows:

$$\text{Level [dB}\mu\text{V]} = 10 \cdot \log(I_{\text{rel}}^2 + Q_{\text{rel}}^2) [\text{dB}] + 20 \cdot \log(\text{RecipGain} / 2^{16}) [\text{dB}] + 0.1 \cdot \text{AntVoltRef} [\text{dB}\mu\text{V}]$$

where I and Q are the real and imaginary parts of each signal sample.

The absolute signal level in [μV] may be calculated as follows:

$$I [\mu\text{V}] = I_{\text{rel}} \cdot (\text{RecipGain} / 2^{16}) \cdot \text{AntVoltLin}$$

$$Q [\mu\text{V}] = Q_{\text{rel}} \cdot (\text{RecipGain} / 2^{16}) \cdot \text{AntVoltLin}$$

$$\text{where AntVoltLin} [\mu\text{V}] = 10^{(0.1 \cdot \text{AntVoltRef}) / 20}$$

Depending on the sample format, as presented in [table 2-5](#), I and Q values can be represented as signed integers on 32-bits (I_{int32}) or 16-bits (I_{int16}) or as 32-bit float values (I_{float}). The relative values of I and Q can be calculated with the following formulas (same applies for Q_{rel}):

- $I_{\text{rel}} = I_{\text{int32}} / (2^{31} - 1)$ where I_{int32} is a signed integer, the most significant bit gives the sign (0 is positive, 1 is negative)
- $I_{\text{rel}} = I_{\text{int16}} / (2^{15} - 1)$ where I_{int16} is a signed integer, the most significant bit gives the sign (0 is positive, 1 is negative)
- $I_{\text{rel}} = I_{\text{float}}$

In the first two cases I_{rel} and Q_{rel} represent relative signal level values between -1 and 1. The absolute signal levels are retrieved through the parameter AntVoltRef as presented above. In the third case, I_{rel} and Q_{rel} can represent directly the absolute signal levels - in this case the RecipGain and AntVoltRef are not used (and are set to RecipGain=1, AntVoltRef=0).

Example

Word position in frame	Frame component name	Hex value	Description
1	uintMagicWord	FB746572	Frame synchronisation
2	uintFrameLength	0000001E	Entire frame length = 30 (in 32-bit units)

Word position in frame	Frame component name	Hex value	Description
3	uintFrameCount	000000FE	Running frame number = 254
4	uintFrameType	00000004	The type of data contained in this frame
5	uintDataHeaderLength	0000000E	Data Header length = 14 (in 32-bit units)
6	uintReserved	00000000	Reserved field
7	uintDatablockCount	00000002	Number of data blocks in this frame = 2
8	uintDatablockLength	00000004	The data block length (in 32-bit units) excluding the data block header = 4. Every data block in this frame will have the same length.
9 10	bigtimeTimeStamp	00035CED 1D63F4D0	Absolute time [μ s] of the first IF signal data sample in this frame
11	uintStatusword	00000000	No status change indications.
12	uintSignalSourceID	00000003	Antenna ID = 3
13	uintSignalSourceState	00000A73	Tuner scan status = position 2675
14	uintTunerFrequency_Low	42E9EC0	Tuner center frequency = 1,123 GHz
15	uintTunerFrequency_High	00000000	
16	uintBandwidth	01312d00	The IF Data bandwidth = 20 MHz
17	uintSamplerate	0493E000	ADC sample rate = 76,8 Msample/s
18	uintInterpolation	00000001	Interpolation factor = none
19	uintDecimation	00000003	Decimation factor referred to the ADC sample rate = 3
20	intAntennaVoltageRef	0000001E	Antenna reference voltage = 3dB μ V
21	uintStatusword	22200000	Beginning of the first Datablock. Statusword contains AGC correction factor = 0.1333 and no flags indications.
22	uintData	23873454	Real part of first sample
23	uintData	34234523	Imaginary part of first sample
24	uintData	56567543	Real part of second sample
25	uintData	34563456	Imaginary part of second sample
26	uintStatusword	41000004	Beginning of the second Datablock. Statusword contains AGC correction factor = 0.2539 and one user flag indication.
27	uintData	45345222	Real part of third sample
28	uintData	546672ab	Imaginary part of third sample
29	uintData	5BB25346	Real part of fourth sample
30	uintData	BBF7673e	Imaginary part of fourth sample

2.2 IF DDCE Data Format

The IF DDCE Data format is used for datastreams containing DDC signal extraction from ESMD tuners.

The IF DDCE Data format is valid for the following frame types:

Table 2-6: IF DDCE Datastream types

Frame type ID	Description	Sample type ID
ekFRH_DATASTREAM_DDCE_IFDATA_32RE_32IM_FIX	Datastream for multi-channel IF Datastreams as used in EMSD-DDCE, 32-bit real- and 32-bit imaginary-part, fixed point, not rescaled.	typIFD_SAMPLE_32RE_32IM_FIX
ekFRH_DATASTREAM_DDCE_IFDATA_16RE_16IM_FIX	Datastream for multi-channel IF Datastreams as used in EMSD-DDCE, 16-bit Real- and 16-bit Imaginary-part, fixed point, not rescaled.	typIFD_SAMPLE_16RE_16IM_FIX

IF DDCE Data Frame Structure

The structure of the IF DDCE Datastream is defined in the `rs_gx40x_ifdata_ddce_header_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

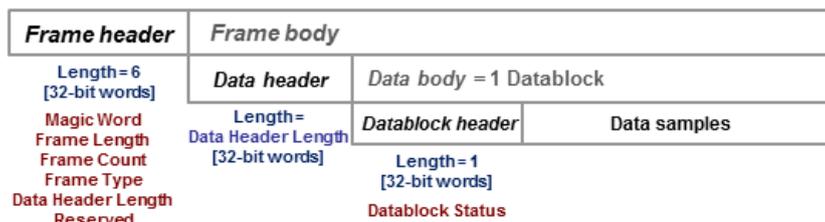


Fig. 2-2: IF DDCE Data frame format

NOTICE

Limited use of the frame structure definition: `typIFD_IFDATA_DDCE_FRAME`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

IF DDCE Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The IF DDCE *Data header* structure, of type `typIFD_IFDATAHEADER_DDCE`, is described in the following table (*Data header* length = 9).

Table 2-7: IF DDCE Data header (typIFD_IFDATAHEADER_DDCE)

Word position in frame	Member name Member type	Description
7	uintSignalSourceID ptypUINT	<p>Signal Source Identifier - this index identifies which channel this frame belongs to.</p> <ul style="list-style-type: none"> Bits 31..30: Signal Source Group ID 0: DDCE instance operated by R&S ESMD-DDCE 1: DDCE instance operated by R&S ESMD-ST Bits 18..29: reserved (0) For statically allocated instances bound to option R&S ESMD-DDCE Bits 7:0: 0...127: channel number For burst emission synthesis, bound to option R&S ESMD-ST Bits 17:0: 0... 262143 : burst emission ID
8	intAntennaVoltageRef ptypINT	<p>Antenna Voltage Reference is the device / parameterization specific correction value for the tuner front-end and the connected antenna. This value typically will not be subject of change on a frame by frame basis. It will rather change due to reconfiguration (in the order of seconds) to the signal processing path.</p> <ul style="list-style-type: none"> Bits #31 to #16: Conversion factor (kFactor) in units 0.1dB/m. These bits are extracted to represent a signed integer number. This factor contains all contributions between air and antenna input connector such as antenna gain, cable attenuation, antenna switch matrix attenuation. (0x8000 if no kFactor is defined). Bits #15 to #0: Rx attenuation value (RxAtt) in units of 0.1dBμV. These bits are extracted to represent a signed integer number. This value expresses all contributions between receiver's antenna input connector and the ADC. <p>The level in dBμV at the receiver's antenna input connector is given by:</p> $\text{Level} = 10 \cdot \log(I^2 + Q^2) + 0.1 \cdot \text{RxAtt}$ <p>The field strength in 0.1dB/m is calculated by:</p> $\text{Field strength} = \text{Level} + 0.1 \cdot \text{kFactor}$
9	intReserved ptypUINT	For future use, should be 0.
10 11	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [μs] - Absolute time of the first IF signal data sample, in the first data block of IF signal data in this frame.
12	uintTunerFrequency_Low ptypUINT	64-bit Tuner Center Frequency [Hz] - least significant 32 bits (uintTunerFrequency_Low) followed by most significant 32 bits (uintTunerFrequency_High)
13	uintTunerFrequency_High ptypUINT	

Word position in frame	Member name Member type	Description
14	fBandwidth ptypFLOAT_SP	IF signal 3dB Bandwidth [Hz].
15	fSamplerate ptypFLOAT_SP	Sample Rate [samples / second].



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

IF DDCE Data Body

The IF Data body consists of only one datablock (of type `typIFD_DATABLOCK_DDCE`) that contains a datablock header `datablockheader_ddceDatablockHeader` (of type: `typIFD_DATABLOCKHEADER_DDCE`) and a datablock body that contains the actual IF samples.

Table 2-8: IF DDCE Datablock header (typIFD_DATABLOCKHEADER_DDCE)

Member name Member type	Description
uintStatusword ptypUINT	<p>Status of the Datablock</p> <ul style="list-style-type: none"> • Bit #31 - Reserved • Bit #30 - dBFS flag <ul style="list-style-type: none"> – 1 indicates that all samples in this frame are considered to be dBFS (dB full scale). – 0 indicates that the values <code>intAntennaVoltageRef</code> and "Reciprocal gain correction" (see Datablock Status of the datablock header) can be used to calculate the corresponding level for each sample. • Bits #29 to #10 - Reserved (must be 0). • Bit #9 - Blanking flag - this flag is set (1) to indicate that the data in this block may have been falsified by some external event, like PTT transmissions that affect the signal quality. • Bit #8 - Invalidity flag - this flag is set (1) to indicate that the data within this block may be corrupt (e.g. the AD converter with which the data was produced was over-range, or the analog signal input from which the data was converted was overloaded), OR any one of the fields in the IF datastream header does not represent the data in this block correctly. • Bits #7 to #0 - User flags for special signaling between IF Data processing components. Set to 0 if not used. <p>Bit 0 : Short-time synthesis end-of-frame flag.</p> <ul style="list-style-type: none"> – 1 : this frame denotes the last frame of a signal synthesis for a particular emission – 0 : there are more frames to come <p>This end-of-frame flag is only valid if <code>SignalSourceGroupID</code> in field <code>uintSignalSourceID</code> of the previous data header carries the value 1.</p>

The Datablock body is defined as an array of size `uintFrameLength - 6 - uintDataHeaderLength` (the size should be ≥ 1) with elements `uintData` of type `ptypUINT`. The actual Data samples have to be extracted from the array. Their structure and size depend on the IF DDCE datastream format (table 2-6). The possible IF data sample formats are given in the table below:

Table 2-9: IF DDCE Data sample format

Sample type ID	Sample format type	Most significant bits	Least significant bits
typIFD_SAMPLE_32RE_32IM_FIX	64-bit I/Q format	First 32-bit Real component	
		Second 32-bit Imaginary component	
typIFD_SAMPLE_16RE_16IM_FIX	32-bit I/Q format	16-bit Imaginary component	16-bit Real component

3 Audio Datastream

The Audio Data format describes the `ekFRH_DATASTREAM__AUDIODATA` datastream type.

Audio Data Frame Structure

The structure of the Audio Datastream is defined in the `rs_gx40x_global_audioformat_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file: `rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

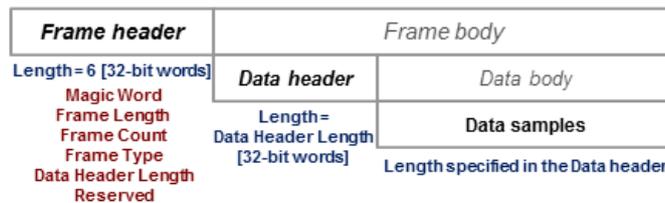


Fig. 3-1: Audio Data frame format

NOTICE

Limited use of the frame structure definition: `typAUDIODATAFRAME`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Audio Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The **basic** *Data header* contains a number of fields, that are always sent.

The **extended** *Data header* contains extra information fields sent after the fields of the basic structure.

The length of the *Data header*, as specified by the `uintDataHeaderLength` parameter from the *Frame header*, gives information about which *Data header* type is used, the basic or the extended one.

The Audio *Data header* structure, of type `typAUDIODATAHEADER`, is described in the following table (*Data header* length = 9).

Table 3-1: Audio Data header (typAUDIODATAHEADER)

Word position in frame	Member name Member type	Description
7	uintSamplerate ptypUINT	Audio Data Sample Rate [Hz].
8	uintStatusword ptypUINT	Status word (bit-coded): <ul style="list-style-type: none"> • Bits #31 to #2 - Reserved • Bit #1 - Squelch status for channel 1 If channel 1 is used <ul style="list-style-type: none"> – 1 indicates over squelch threshold – 0 indicates under squelch threshold If channel 1 is not used, the value is always 0 • Bit #0 - Squelch status for channel 0. <ul style="list-style-type: none"> – 1 indicates over squelch threshold – 0 indicates under squelch threshold
9	uintCenterFrequency_Low ptypUINT	64-bit Center Frequency of demodulator [Hz] - least significant 32 bits (<code>uintCenterFrequency_Low</code>) followed by most significant 32 bits (<code>uintCenterFrequency_High</code>)
10	uintCenterFrequency_High ptypUINT	
11	uintDemodBandwidth ptypUINT	Demodulation Bandwidth [Hz].
12	eDemodulationType typAUDIODATA_DEMOD	Demodulation Type - demodulation used to produce the audio data stream. See table below.
13	uintSampleCount ptypUINT	Audio Data Sample Count per channel
14	uintChannelCount ptypUINT	Audio Data Channel Count . Not limited. Usually one or two channels are used.
15	uintSampleSize ptypUINT	Audio Data Sample Size [bytes]. Because the basic data format is defined in 32-bit words, the audio datastream samples can be represented on 1, 2 or 4 bytes.

The demodulation methods (enumeration type: `eAUDIODATA_DEMOD`) that can be used when producing the Audio datastream are presented in the following table (in the future, it is possible that other demodulation methods are added to this list):

Table 3-2: Demodulation methods

Member name	Value	Description
<code>ekAUDIODATA_DEMOD_FM</code>	0	Frequency demodulation
<code>ekAUDIODATA_DEMOD_AM</code>	1	Amplitude demodulation

Member name	Value	Description
ekAUDIODATA_DEMOD_ISB	5	Independent Side Band (ISB) sidebands (channels). In practice two datastream channels are generated, channel 1 for LSB and channel 0 for USB
ekAUDIODATA_DEMOD_CW	6	Continuous Wave
ekAUDIODATA_DEMOD_USB	7	Upper Sideband
ekAUDIODATA_DEMOD_LSB	8	Lower Sideband
ekAUDIODATA_DEMOD_DIGITAL	0x100	Digital demodulation
ekAUDIODATA_DEMOD_UNKNOWN	0xFFFFFFFF	Unknown

The Extended Audio *Data header* structure, of type `typAUDIODATAHEADER_EX`, is described in the following table (total Data header length = 11 [32-bit words]).

Table 3-3: Extended Audio Data header (`typAUDIODATAHEADER_EX`) - extra fields only

Word position in frame	Member name Member type	Description
16	<code>bigtimeTimeStamp</code>	64-bit Timestamp [μs] - Absolute time of the first data sample in this frame
17	<code>ptypBIGTIME</code>	



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Audio Data Body

The Audio Data body contains the actual Audio Data samples. The samples, PCM coded signed integers, are sent in packed form (no alignment gaps) as an array of unsigned integers (`uintData` are of type `ptypUINT`). In case of two or more channels, the datastreams are interleaved, starting with channel 0.

Table 3-4: Audio Data 32-bit sample formats

Word position in data body	Data body - Audio Mono (one channel)	Data body - Audio Stereo (two channels)
	<i>Bits #31 to #0</i>	<i>Bits #31 to #0</i>
1	Sample #0	Sample #0, Channel 0
2	Sample #1	Sample #0, Channel 1
3	Sample #2	Sample #1, Channel 0
4	Sample #3	Sample #1, Channel 1

Table 3-5: Audio Data 16-bit sample formats

Word position in data body	Data body - Audio Mono (one channel)		Data body - Audio Stereo (two channels)	
	Bits #31 to #16	Bits #15 to #0	Bits #31 to #16	Bits #15 to #0
1	Sample #1	Sample #0	Sample #0, Channel 1	Sample #0, Channel 0
2	Sample #3	Sample #2	Sample #1, Channel 1	Sample #1, Channel 0

Table 3-6: Audio Data 8-bit sample formats

Word position in data body	Data body - Audio Mono (one channel)				Data body - Audio Stereo (two channels)			
	Bits #31 to #24	Bits #23 to #16	Bits #15 to #8	Bits #7 to #0	Bits #31 to #24	Bits #23 to #16	Bits #15 to #8	Bits #7 to #0
1	Sample #3	Sample #2	Sample #1	Sample #0	Sample #1 Channel 1	Sample #1 Channel 0	Sample #0 Channel 1	Sample #0 Channel 0
2	Sample #7	Sample #6	Sample #5	Sample #4	Sample #3 Channel 1	Sample #3 Channel 0	Sample #2 Channel 1	Sample #2 Channel 0

The data body size is given by: $SampleCount * ChannelCount * SampleSize$ [bytes] plus padding bits to a multiple of 32 bits.

4 Tuner Datastreams

4.1 Scan Data Format

The Scan Data format is used for transmissions of scan data containing tuning level indicator and frequency tuning offset. The Scan Data format is valid for the following datastream types:

Table 4-1: Scan Datastream types

Datastream type ID	Description	Sample parameters
ekFRH_DATASTREAM__SCAN__LEVEL	Datastream of scan data containing tuning level indicator information.	nLevel
ekFRH_DATASTREAM__SCAN__TUNING	Datastream of scan data containing frequency tuning offset information.	nTuning
ekFRH_DATASTREAM__SCAN__LEVEL_TUNING	Datastream of scan data containing tuning level indicator and frequency tuning offset information.	nLevel nTuning

For the above datastream types, the same frame structure is used, the only difference is the payload carried by the datablock body (as described in the table above).

Scan Data Frame Structure

The structure of the Scan Datastream is defined in the `rs_gx40x_scandata_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

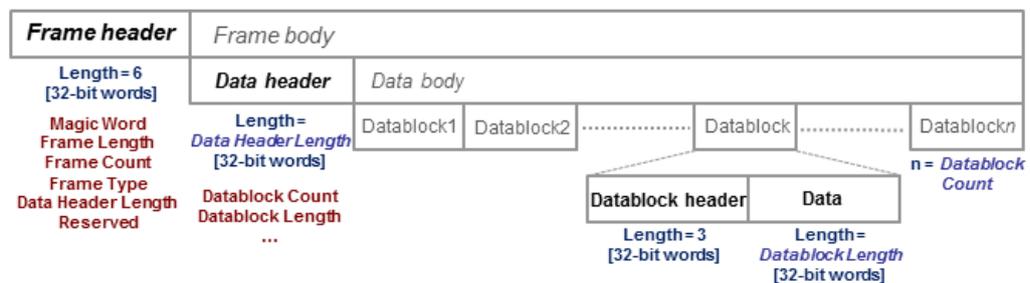


Fig. 4-1: Scan Data frame format

NOTICE**Limited use of the frame structure definition: `typSCANDATA_FRAME__LEVEL`, `typSCANDATA_FRAME__TUNING`, `typSCANDATA_FRAME__LEVEL_TUNING`**

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Scan Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The *Scan Data header* structure, of type `typSCANDATA_HEADER`, is described in the following table (*Data header* length = 5).

Table 4-2: Scan DATA header (`typSCANDATA_HEADER`)

Word position in frame	Member name Member type	Description
7	nDatablockCount ptypUINT	Datablock Count - represents the number of scan data blocks (payload) in this frame. The first two fields determine the granularity of the interface.
8	nDatablockLength ptypUINT	Datablock Length - The number 32-bit words in each data block body (excluding the data block header). The size of the payload is constant for each frame type.
9 10	nTimeStamp ptypBIGTIME	Timestamp of this frame in microseconds. Absolute time of the first data sample, in the first data block in this frame.
11	nReserved2 ptypUINT	Reserved



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Scan Data Body

The *Scan Data body* contains the one or more scan results arranged as an array of `nDatablockCount` datablocks (the actual stream payload).

There are three datablock types, depending on the carried payload:

- `typSCANDATABLOCK__LEVEL`
- `typSCANDATABLOCK__TUNING`
- `typSCANDATABLOCK__LEVEL_TUNING`

Each datablock has a datablock header (common for all datablock types): `scandatablock_header` (of type `typSCANDATABLOCK_HEADER`) and a datablock body that contains the actual data:

- `scandatablock_body__level` of type `typSCANDATABLOCK_BODY__LEVEL`
- `scandatablock_body__tuning` of type `typSCANDATABLOCK_BODY__TUNING`
- `scandatablock_body__level_tuning` of type `typSCANDATABLOCK_BODY__LEVEL_TUNING`

Table 4-3: Scan Datablock header (`typSCANDATABLOCK_HEADER`)

Member name Member type	Description
<code>nTunerFreqLow</code> <code>ptypUINT</code>	Least significant 32 bits of the 64-bit representation Tuner Center Frequency [Hz] at which the following scan result was measured. A scan re-start is indicated by the special value: <code>kSCANDATA_WRAPAROUND_FLAG__FREQUENCY_LOW</code>
<code>nTunerFreqHigh</code> <code>ptypUINT</code>	Most significant 32 bits of the 64-bit representation Tuner Center Frequency [Hz] at which the following scan result was measured. A scan re-start is indicated by the special value: <code>kSCANDATA_WRAPAROUND_FLAG__FREQUENCY_HIGH</code>
<code>nChannelNumber</code> <code>ptypUINT</code>	Scan Channel Number at which the following scan result was measured. It represents the index in the predefined frequency list in case of a Memory Scan, and the increment (starting with 0) in case of a Frequency Scan (the scanned frequency is given by: <code>start_frequency + increment * frequency_step</code>). A scan re-start is indicated by the special value: <code>kSCANDATA_WRAPAROUND_FLAG__CHANNEL</code>

The datablock body is an array of size = `uintDatablockLength`. The array elements depend on the datablock type:

Table 4-4: Scan Data sample format

Datablock type	Member name Member type	Description
<code>typSCANDATABLOCK_BODY__LEVEL</code>	<code>nLevel</code> <code>ptypINT</code>	Tuning level indicator [0.1 dB μ V]. A scan re-start is indicated by the special value: <code>kSCANDATA_WRAPAROUND_FLAG__LEVEL</code>
<code>typSCANDATABLOCK_BODY__TUNING</code>	<code>nTuning</code> <code>ptypINT</code>	The offset [Hz] from the tuner frequency to the detected maximum within this spectrum segment. A scan re-start is indicated by the special value: <code>kSCANDATA_WRAPAROUND_FLAG__TUNING</code>
<code>typSCANDATABLOCK_BODY__LEVEL_TUNING</code>	<code>nLevel</code> <code>ptypINT</code>	same as above
	<code>nTuning</code> <code>ptypINT</code>	same as above



The special values are defined in `rs_gx40x_scandata_if_defs.h`

4.2 Signal Level Indicator Data Format

The Signal Level Indicator Data format is used for individual Signal Level Indications of the signal received by the tuner. These indications are periodically generated with a maximal rate of 10 per second. This data format describes the `ekFRH_DATASTREAM__LEVELDATA` datastream type.

Signal Level Indicator Data Frame Structure

The structure of the Signal Level Indicator Datastream is defined in the `rs_gx40x_leveldata_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The Frame body has a fixed size and contains the Signal Level Indicator data values directly after the Frame header. A Data header is not used.

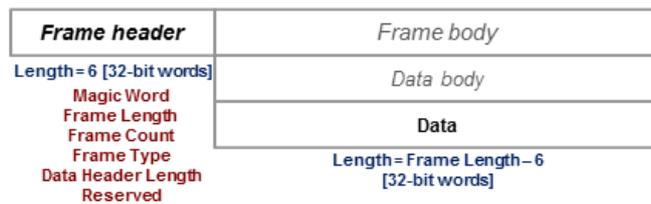


Fig. 4-2: Signal Level Indicator Data frame format

NOTICE

Limited use of the frame structure definition: `typLEVELDATA_FRAME`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Signal Level Indicator Data Body

The Signal Level Indicator data body of type `typLEVELDATA`, contains the actual Signal Level Indicator data values.

Table 4-5: Signal Level Indicator Data (typLEVELDATA)

Word position in frame	Member name Member type	Description
7	uintStatusword ptypUINT	Signal Level indicator Status (bit-coded): <ul style="list-style-type: none"> • Bit #31 - Overload flag: <ul style="list-style-type: none"> – 0 - no overload – 1 - Tuner ADC overloaded (input signal too strong) • Bit #30 - RF_MUTE flag <ul style="list-style-type: none"> – 0 - RF_MUTE not active – 1 - RF_MUTE active (signal disconnected from tuner) • Bits #29 to #0 - Reserved
8	intTuningLevel ptypINT	Signal Level [dB μ V] - the level of the tuner input signal
9	intRelativeLevelADC ptypINT	Relative signal level [0.1 dB-full-scale] in relation to the ADC input range (0 dB represents the full scale, i.e. the input signal covers the full input range of the ADC).
10	uintCurrentAttenuation ptypUINT	Current attenuation [dB]

4.3 Tuner PIF Panorama Data Format

The Tuner PIF Panorama Data format is used for polychrome PIF Panorama visualization of the tuner spectrum. This data format corresponds to the `ekFRH_DATASTREAM__TUNER_PIFPAN_DATA` datastream type (the corresponding "Frame Type" value can be found in the `rs_gx40x_global_frame_types_if_defs.h`).

Tuner PIF Panorama Data frame structure

The Tuner PIF Panorama Data frame consists of the global frame header of type `typFRH_FRAMEHEADER`, as described in "[Global Frame header](#)" on page 5, followed by the frame payload (direct from tuner in EB200-Datagram format).

See the tuner documentation for frame payload content (EB200-Datagram format). The content is supplemented with padding bytes to a multiple of 32-bits.

4.4 Tuner HF (EM010) Data Formats

4.4.1 EM010 Tuning Indicator Data Format

The Tuning Indicator Data format is used for Receiver Tuning Indicator and Status information. This data format describes the `ekFRH_DATASTREAM__HF_TUNING_INDICATOR_DATA` datastream type.

Tuning Indicator Data Frame Structure

The structure of the Tuning Indicator Datastream is defined in the `rs_gx40x_leveldata_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_em010_hf_if_defs.h`.

The Frame body contains one Datablock directly after the Frame header. A Data header is not used.

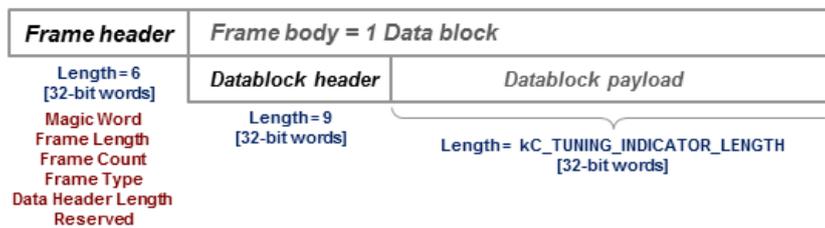


Fig. 4-3: Tuning Indicator Data frame format

NOTICE

Limited use of the frame structure definition: `typFRAME_EM010_HF_TUNING_INDICATOR_AND_STATUS_DATA`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Tuning Indicator Data Block

The Tuning Indicator data body contains one datablock of type `typEM010_HF_TUNING_INDICATOR_AND_STATUS_DATA_BLOCK`.

Table 4-6: Tuning Indicator Data (*typEM010_HF_TUNING_INDICATOR_AND_STATUS_DATA_BLOCK*)

Word position in frame	Member name Member type	Description
7	unReserved ptypUINT	Reserved
8 9	bigtimeSignalTime ptypBIGTIME	Timestamp of this Tuning Indicator and Status package in [μ s] (see the "EM010 HF Receiver" manual: TUNING_INDICATOR_DATA – <i>signal-time</i>)
10	unFrequency ptypUINT	Frequency [Hz] of tuner (see the "EM010 HF Receiver" manual: TUNING_INDICATOR_DATA – <i>frequency</i>)
11	nSignalLevel ptypINT	Signal Level at antenna input in [dB μ V] - valid for the Tuning Indicator data in this package (see the "EM010 HF Receiver" manual: TUNING_INDICATOR_DATA – <i>signallevel</i>)
12	bOverloadFlag ptypBOOL	Condition of the Overload flag - When <code>TRUE</code> the Antenna input is automatically disabled for 1 second. (see the "EM010 HF Receiver" manual: TUNING_INDICATOR_DATA – <i>overloadflag</i>)
13	bHFmuteFlag ptypBOOL	Condition of the HF-Mute flag - Mute can be set via the input connector of the EM010 front panel (see the "EM010 HF Receiver" manual: TUNING_INDICATOR_DATA – <i>hfmuteflag</i>)
14	bSquelchFlag ptypBOOL	Condition of the Squelch flag (see the "EM010 HF Receiver" manual: TUNING_INDICATOR_DATA – <i>squelchflag</i>)
15	unReserved_2 ptypUINT	Reserved
16 ... 36	unDatablock ptypUINT	The Tuning Indicator data (FFT) (defined as an array of ptypUINT elements with size given by the constant <code>KC_TUNING_INDICATOR_LENGTH = 21</code> (see the "EM010 HF Receiver" manual: TUNING_INDICATOR_DATA – <i>datablock</i>))

4.4.2 EM010 Scan Channel Found (SCF) and EM010 Scan Frequency Found (SFF) Data Formats

The SCF and SFF Scan Data formats describe the `ekFRH_DATASTREAM__HF_SCF_DATA` and `ekFRH_DATASTREAM__HF_SFF_DATA`, respectively, datastream types.

SCF and SFF Scan Data Frame Structure

The structure of the SCF and SFF Scan Datastreams are defined in the `rs_gx40x_em010_hf_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file: `rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

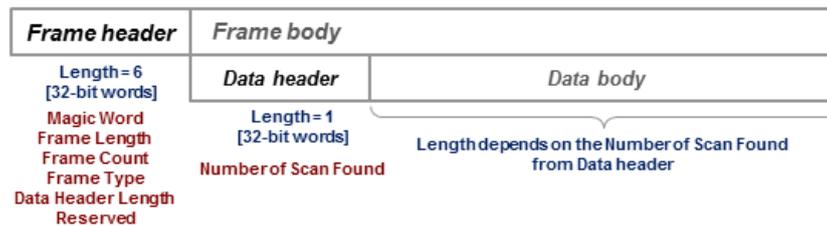


Fig. 4-4: SCF and SFF Scan Data frame format

NOTICE

Limited use of the frame structure definition: typ-FRAME_EM010_HF_SCAN_CHANNEL_FOUND, typ-FRAME_EM010_HF_SCAN_FREQUENCY_FOUND

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

SCF and SFF Scan Data header

The *Scan Data header* contains the number of Data samples contained in this frame.

The *Scan Data header* structure, of type `typEM010_HF_SCAN_DATA_HEADER`, is described in the following table (*Data header* length = 1).

Table 4-7: SCF and SFF Scan Data header (`typEM010_HF_SCAN_DATA_HEADER`)

Word position in frame	Member name Member type	Description
7	<code>unNumberOfScanFound</code> pstdint	The Number of scan found packets



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

SCF and SFF Scan Data body

The SCF and SFF Scan Data body contain one or more scan data blocks with scan results, defined as an structure of type `typEM010_HF_SCAN_CHANNEL_FOUND` for channel scan and `typEM010_HF_SCAN_FREQUENCY_FOUND` for frequency scan. The

number of scan results listed in the data body is given in the Data header element:
Number of scan found packets.

Table 4-8: SCF Scan Data body element (typEM010_HF_SCAN_CHANNEL_FOUND)

Word offset	Member name Member type	Description
0	unFoundChannel ptypUINT	Channel (see the "EM010 HF Receiver" manual: CHANNEL_FOUND_DATA – <i>channel</i>)
1	nSignallevel ptypINT	Signal level in [dB μ V] of the found channel (see the "EM010 HF Receiver" manual: CHANNEL_FOUND_DATA – <i>signallevel</i>)

Table 4-9: SFF Scan Data body element (typEM010_HF_SCAN_FREQUENCY_FOUND)

Word offset	Member name Member type	Description
0	unFoundFrequency ptypUINT	Frequency of detected emission (see the "EM010 HF Receiver" manual: FREQUENCY_FOUND_DATA – <i>frequency</i>)
1	nSignallevel ptypINT	Signal level in [dB μ V] of the found emission (see the "EM010 HF Receiver" manual: FREQUENCY_FOUND_DATA – <i>signallevel</i>)

4.4.3 EM010 Scan Sweep Restarted (SSR) Data Format

The SSR Status Data format is used for Status Data Indication: Scan Sweep Restarted of the EM010 receiver. The indication that a new scan is started is used for both Frequency and Channel Scans. This data format describes the `ekFRH_DATASTREAM__HF_SSR_DATA` datastream type.

SSR Data Frame Structure

The structure of the SSR Status Datastream is defined in the `rs_gx40x_em010_hf_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The Frame body consists of only one word indicating that the EM010 has reached the end of the configured scan and is starting again. A Data header is not used.

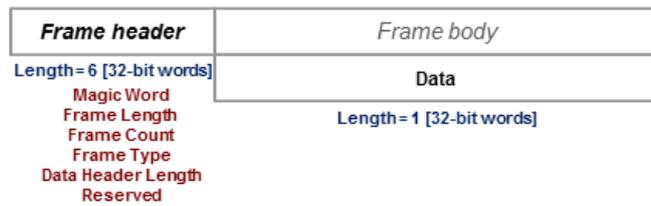


Fig. 4-5: SSR Data frame format

NOTICE

Limited use of the frame structure definition: typ- FRAME_EM010_HF_SCAN_RESTARTED

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

SSR Data Body

The SSR data body consists of only one word: `unScanRestartedCycleCount` of type `ptypUINT` that gives the number of times the (configured) scan has completed. For further details see: "EM010 HF Receiver" manual: SWEEP_RESTARTED_DATA.

5 Spectrum Datastreams

5.1 Spectrum Data Format

The Spectrum Data format (32-bit float) describes the `ekFRH_DATASTREAM__SPECDATA_16BIT` and `ekFRH_DATASTREAM__SPECDATA_FLOAT` datastream types.

Spectrum Data Frame Structure

The structure of the Spectrum Datastream is defined in the `rs_gx40x_specdata_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

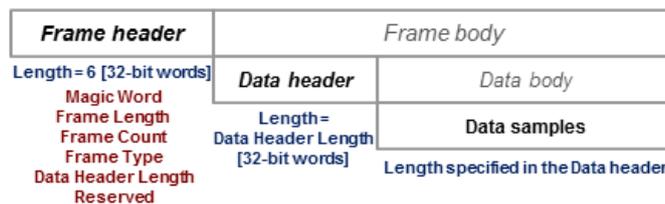


Fig. 5-1: Spectrum Data frame format

NOTICE

Limited use of the frame structure definition: `typSPECDATA_FLOAT`, `typSPEC- DATA_16BIT`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Spectrum Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The **basic Data header** contains a number of fields, that are always sent.

The **extended Data header** contains extra information fields sent after the fields of the basic structure.

The length of the *Data header*, as specified by the `uintDataHeaderLength` parameter from the *Frame header*, gives information about which *Data header* type is used, the basic or the extended one.

The Spectrum *Data header* structure, of type `typSPECDATA_HEADER`, is described in the following table (*Data header* length = 10).

Table 5-1: Spectrum Data header (typSPECDATA_HEADER)

Word position in frame	Member name Member type	Description
7 8	bigtimeTimeStamp ptypBIGTIME	Timestamp in microseconds - Absolute time of the first data sample in this frame
9 10	uintCenterFrequency_Low uintCenterFrequency_High ptypUINT	64-bit Center Frequency for spectrum calculation [Hz] - least significant 32 bits (<code>uintCenterFrequency_Low</code>) followed by most significant 32 bits (<code>uintCenterFrequency_High</code>)
11	uintSamplerate ptypUINT	Sample Rate [Hz] of the data from which the spectrum was calculated.
12	uintFFTLength ptypUINT	FFT length - Number of points (bins) in the Fast Fourier Transform (FFT) window.
13	uintStatusword ptypUINT	<p>Status word (bit-coded):</p> <ul style="list-style-type: none"> • Bits #31 to #18 - Reserved, must be 0 • Bit #17 - Fragment Flag indicating that this is a fragment of a complete spectrum and the bins between <code>uintLeftDispInterval</code> and <code>uintRightDispInterval</code> should be overwritten. All other bins shall stay unaffected. If the flag is set, the bandwidth is equal to the sample rate. • Bit #16 - Blanking Flag indicating that the spectrum has been calculated using data that was flagged as blanking • Bits #15 to #12 - Sample source used for FFT calculation. The available types are listed in the <code>eSPECDATA_SAMPLESOURCE</code> enumeration. • Bit #11 - Reserved • Bit #10 - dBFs Flag. <ul style="list-style-type: none"> – 0 indicates that the level is calculated using the <code>ReferenceValue</code> – 1 indicates that there is no level information available and dB full scale was used • Bit #9 - Invalidity Flag indicating that the spectrum has been calculated using data that was flagged as invalid • Bit #8 - Level Type Flag indicating the mode with which the spectral data level was calculated (0 = linear, 1 = logarithmic, as described in the <code>eSPECDATA_LEVELINFO</code> enumeration) • Bits #7 to #4 - Window type of the spectral data. The available types are listed in the <code>eSPECDATA_WINTYPES</code> enumeration. • Bits #3 to #0 - Display mode of the spectral data. The available types are listed in the <code>eSPECDATA_DISPLAY_MODES</code> enumeration.

Word position in frame	Member name Member type	Description
14	float_spReferenceValue ptypFLOAT_SP	Level reference value. Given in [μ V] or full scale for linear calculation mode and [dBm] or full scale dBFS for logarithmic calculation mode.
15	uintLeftDispInterval ptypUINT	The index (starting at 0) of the leftmost bin (FFT point) to be sent (for display): D_L
16	uintRightDispInterval ptypUINT	The index of the rightmost bin (FFT point) to be sent (for display) D_R . Following relation applies: $0 \leq D_L \leq D_R < \text{FFTLength}$

The Extended Spectrum *Data header* structure, of type `typSPECDATA_HEADER_EXTENDED`, is described in the following table (total Data header length = 12 [32-bit words]).

Table 5-2: Extended Spectrum Data header (typSPECDATA_HEADER_EXTENDED) - extra fields only

Word position in frame	Member name Member type	Description
18	intKFactor ptypINT	kFactor of the current antenna to determine field strength in 0.1dB/m. The predefined <code>kK_FACTOR_NOT_VALID</code> value is used if no kFactor is defined
19	uintSampleRate_High ptypUINT	Sample rate of the data from which the spectrum was calculated -- in samples/second [Hz] - most significant 32 bits of the 64-bit representation



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Spectrum Data Body

The Spectrum *Data body* contains the actual Spectrum Data samples arranged as an array of elements. The number and type of elements depends on the Spectrum data-stream type as described in the following table:

Table 5-3: Spectrum Data sample format

Datastream type	Data sample type	Data body array structure
<code>ekFRH_DATASTREAM__SPECDATA_FLOAT</code>	ptypFLOAT_SP	Data body contains $(\text{uintRightDispInterval} - \text{uintLeftDispInterval} + 1)$ floating point elements
<code>ekFRH_DATASTREAM__SPECDATA_16BIT</code>	ptypUINT	Data body contains $\text{ceil}((\text{uintRightDispInterval} - \text{uintLeftDispInterval} + 1) / 2)$ value-pair elements. Each value-pair contains two 16-bit signed fractional values. Values with even indices are placed in the most significant 16 bits. Values with odd indices are placed in the least significant 16 bits. If the number of values is odd, the last value (stuffing value) is set to zero

5.2 Segmentation Spectrum Data Format

The Segmentation Spectrum Data format (32-bit float) describes the `ekFRH_DATASTREAM__SEGMENTATION_SPECDATA_FLOAT` datastream type.

Segmentation Spectrum Data Frame Structure

The structure of the Segmentation Spectrum Datastream is defined in the `rs_gx40x_segmentation_specdata_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

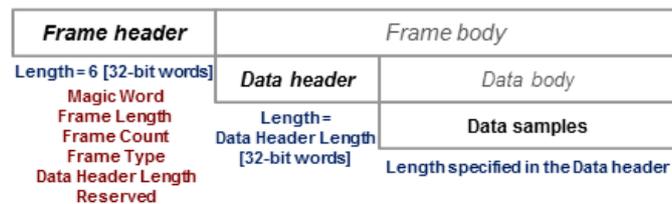


Fig. 5-2: Segmentation Spectrum Data frame format

NOTICE

Limited use of the frame structure definition: `typSEGMENTATION_SPECDATA_FLOAT`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Segmentation Spectrum Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common properties of the Data samples.

The Segmentation Spectrum *Data header* structure, of type `typSEGMENTATION_SPECDATA_HEADER`, is described in the following table (total Data header length = 12 [32-bit words]).

**Table 5-4: Segmentation Spectrum Data header structure (typSEGMENTATION_SPEC-
DATA_HEADER)**

Word position in frame	Member name Member type	Description
7 to 16	specdataheaderHeader typSPECDATA_HEADER	Spectrum data specific header as described at table 5-1
17	uintNumberOfSegments ptypUINT	The number of segments found
18	uintIndexOfCenterSegment ptypUINT	The index number (starting with 0) of the center segment



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Segmentation Spectrum Data Body

The Segmentation Spectrum Data body consists of two Data blocks. The first Data block contains the actual Spectrum Data samples structured as an array of floating point elements. The second Data block contains the boundaries of the found segments (array of segments defined as a structure of type: typSEGMENT_BOUNDARY).

Table 5-5: Segmentation Spectrum Data Body format

Member name Member type	Description
float_spFFTBin ptyp FLOAT_SP	Samples Data block - array containing (uintRightDisplInterval-uintLeftDisplInterval+1) floating point values
segmentboundaryBoundaries typSEGMENT_BOUNDARY	Segments Data block - array containing <code>uintNumberOfSegments</code> elements of type <code>typSEGMENT_BOUNDARY</code> defining the boundaries of the found segments (left/right FFT-Bin and corresponding left/right absolute frequencies)
uintLeftSegmentFFTBin ptypUINT	Left FFT bin - the number (starting at 0) of the leftmost bin (FFT point) of this segment
uintRightSegmentFFTBin ptypUINT	Right FFT bin - the number of the rightmost bin (FFT point) of this segment
uintLeftSegmentFrequency_Low uintLeftSegmentFrequency_High ptypUINT	Left boundary frequency - of the segment in [Hz] - 64-bit representation: least significant 32 bits (<code>uintLeftSegmentFrequency_Low</code>) followed by most significant 32 bits (<code>uintLeftSegmentFrequency_High</code>)
uintRightSegmentFrequency_Low uintRightSegmentFrequency_High ptypUINT	Right boundary frequency - of the segment in [Hz] - 64-bit representation: least significant 32 bits (<code>uintRightSegmentFrequency_Low</code>) followed by most significant 32 bits (<code>uintRightSegmentFrequency_High</code>)

6 Symbol Datastreams

The Symbol Data format is used for demodulation result data (symbol stream) transmissions. It describes the `ekFRH_DATASTREAM__SYMBOLDATA` datastream type.

Symbol Data Frame Structure

The structure of the Symbol Datastream is defined in the `rs_gx40x_symboldata_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

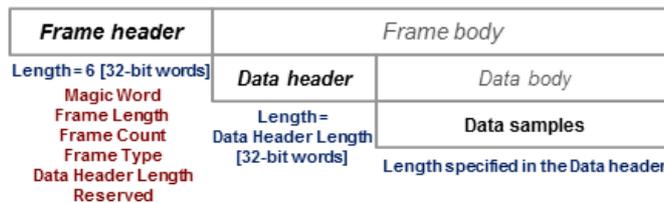


Fig. 6-1: Symbol Data frame format

NOTICE

Limited use of the frame structure definition: `typSYMBOLDATAFRAME`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Symbol Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The Symbol *Data header* structure, of type `typSYMBOLDATAHEADER`, is described in the following table (*Data header* length = 12).

Table 6-1: Symbol Data header (typSYMBOLDATAHEADER)

Word position in frame	Member name Member type	Description
7 8	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [μ s] - Absolute time of the first sample of the data in the frame from which the symbols were calculated.
9 10	uintCenterFrequency_Low uintCenterFrequency_High ptypUINT	64-bit Center Frequency of demodulator [Hz] - least significant 32 bits (uintCenterFrequency_Low) followed by most significant 32 bits (uintCenterFrequency_High)
11	float_spFrequencyDeviation ptypFLOAT_SP	Frequency deviation Deviation from tuner center frequency due to automatic frequency control unit [Hz].
12 13	packedcharModulationType ptypPACKEDCHAR	Modulation type: Token defining the modulation type (defined as a string of ptypPACKEDCHAR characters with size = 2, as defined by the kSYMBOLDATA_MODTYPE_STRING_LEN constant). The modulation type is described through 8 ASCII (ANSI X3.4) characters (empty fields are filled with \0). Note: modulation types with 8 characters have no terminating \0.
14	uintStatusword ptypUINT	Status word (bit-coded): <ul style="list-style-type: none"> • Bits #31 to #4 - Reserved (must be set to 0) • Bit #3 - indicates Morse data (bits #2 to #0 have to be set to 0). In case of morse data, another format of the symbol data and Soft-decision value will be used. If channel 1 is not used, the value is always 0 • Bit #2... #0 - Soft decision type. Defines the type of the Soft-decision value from the Symbol data. <ul style="list-style-type: none"> – 0 indicates a 1x16-bit fractional signed real value in the interval [-1.0 to 1.0] – 1 indicates a 2x8-bit fractional signed complex value in the interval [-1.0 to 1.0]. These two numbers represent a complex number with a real and an imaginary part (the imaginary part represents the most significant 8 bits and the real part the least significant 8 bits)
15	uintChannelCount ptypUINT	Number of channels (larger than one only in multi-channel mode)
16	float_spSymbolRate ptypFLOAT_SP	Symbol rate in symbols / second. This value is derived as an average for all the symbols in this symbol-data frame. Values between 1 and 4800 are permitted.
17	uintSymbolValency ptypUINT	Symbol valency: number of possible different symbol values for the given modulation type (i.e. maximum numeric value a symbol may have). For all symbol values we have: $0 \leq \text{symbol value} \leq \text{Symbol valency} \leq 128$.
18	uintSymbolCount ptypUINT	Symbol count: number of symbols in this frame (to follow this header)



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Symbol Data Body

The Symbol Data body contains the actual Symbol Data samples. The samples are sent as an array of unsigned integers (of type `ptypUINT`). The data body size is given by: `Symbol count` [32-bit words].

Each 32-bit data sample has the following structure:

- Bits #31 to #16 - **Soft decision** value (the format of the value is given by the `Soft decision type` from the `Status` word in the *Data header*. This value depends on the modulation type. In case of Morse data, this value represents the duration of the tone / pause in 100µs steps.
- Bits #15 to #13 - Reserved (have to be set to 0)
- Bit #12 - **Burst start flag** denotes the first symbol in a demodulated signal burst. For multi channel mode, for each channel the first symbol in a demodulated signal burst carries this flag.
- Bits #11 to #8 - **Symbol quality** (confidence value) expressed as an unsigned integer (depends on the modulation type). Higher values indicate higher quality (confidence)
- Bit #7 - **Channel 0 flag**: the value 1 indicates that this symbol belongs to channel 0. For multi channel mode, the datastreams are interleaved, starting with channel 0 in increasing order of the center frequency. Thus, the symbol following a channel-0 symbol belongs to channel-1 and so on. In case of single channel mode, this flag is always set to 1.
- Bits #6 to #0 - The actual **symbol value** (for some modulation types there is a specific symbol to bit pattern mapping). The symbol value is represented on 7 bits. Depending on the method, only a certain number of those bits are valid as specified by the Symbol valency (for example, with FSK-2 only one, the least significant bit is used).

Table 6-2: Mapping symbol value to Morse data

Morse data	Binary value	Hex value	Constant defined in header file
DOT tone	00000000	0x0	KSYMBOLDATA_MORSE__DOT
DASH tone	00000001	0x1	KSYMBOLDATA_MORSE__DASH
inter-tone gap	00000010	0x2	KSYMBOLDATA_MORSE__SHORT_PAUSE
inter-character gap	00000011	0x3	KSYMBOLDATA_MORSE__MEDIUM_PAUSE
inter-word gap	00000100	0x4	KSYMBOLDATA_MORSE__LONG_PAUSE
new line	00000101	0x5	KSYMBOLDATA_MORSE__NEW_LINE
Error messages			
tone too short	10000000	0x40	KSYMBOLDATA_MORSE__DOT_TOO_SHORT
tone too long	10000001	0x41	KSYMBOLDATA_MORSE__DASH_TOO_LONG
gap too short	10000010	0x42	KSYMBOLDATA_MORSE__PAUSE_TOO_SHORT

7 Time Domain Datastreams

7.1 Time Domain Data Format

The Time Domain Data format describes the `ekFRH_DATASTREAM__TIMEDOMAIN_DATA` datastream type.

Time Domain Data Frame Structure

The structure of the Time Domain Datastream is defined in the `rs_gx40x_timedomain_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file: `rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

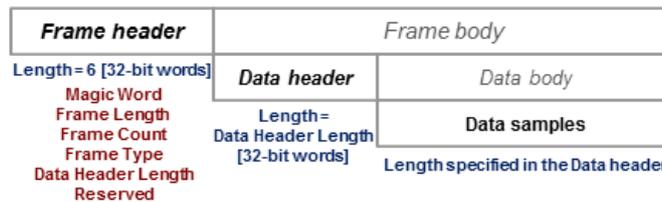


Fig. 7-1: Time Domain Data frame format

NOTICE

Limited use of the frame structure definition: `typTIMEDOMAINDATAFRAME`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Time Domain Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The Time Domain *Data header* structure, of type `typTIMEDOMAINDATAHEADER`, is described in the following table (*Data header* length = 7).

Table 7-1: Time Domain Data header (typTIMEDOMAINDATAHEADER)

Word position in frame	Member name Member type	Description
7 8	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [μ s] - Absolute time of the first sample of the data in the frame from which the time domain data was calculated.
9 10	uintTunerFrequency_Low ptypUINT uintTunerFrequency_High ptypUINT	64-bit Tuner Frequency of demodulator [Hz] - least significant 32 bits (uintTunerFrequency_Low) followed by most significant 32 bits (uintTunerFrequency_High)
11	uintStatusword ptypUINT	Status word (bit-coded): <ul style="list-style-type: none"> Bits #31 to #0 - Reserved (must be set to 0)
12	float_spSampleRate ptypFLOAT_SP	Sample rate
13	uintSampleCount ptypUINT	Sample count: number of symbols to follow this header



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Time Domain Data Body

The Time Domain Data body contains the actual Time Domain Data samples. The samples are sent as an array of type `ptypFLOAT_SP`. The data body size is given by: `Sample count [32-bit words]`.

7.2 Instantaneous Data Format

The Instantaneous Data format describes the `eKFRH_DATASTREAM__INSTANTANEOUSDATA` datastream type.

Instantaneous Data Frame Structure

The structure of the Instantaneous Data stream is defined in the `rs_gx40x_instantaneousdata_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

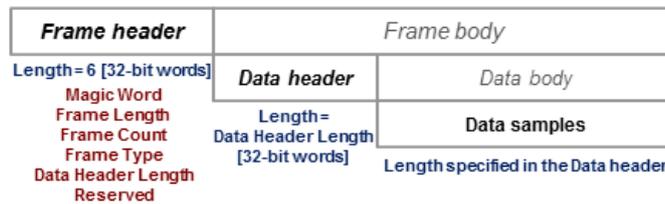


Fig. 7-2: Instantaneous Data frame format

NOTICE

Limited use of the frame structure definition: `typINSTANTANEOUSDATAFRAME`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Instantaneous Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The Instantaneous *Data header* structure, of type `typINSTANTANEOUSDATAHEADER`, is described in the following table (*Data header* length = 8).

Table 7-2: Instantaneous Data header (`typINSTANTANEOUSDATAHEADER`)

Word position in frame	Member name Member type	Description
7 8	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [μs] - Absolute time of the first sample of the data in the frame from which the instantaneous data was calculated.
9 10	uintTunerFrequency_Low ptypUINT uintTunerFrequency_High ptypUINT	64-bit Tuner Frequency of demodulator [Hz] - least significant 32 bits (<code>uintTunerFrequency_Low</code>) followed by most significant 32 bits (<code>uintTunerFrequency_High</code>)
11	uintStatusword ptypUINT	Status word (bit-coded): <ul style="list-style-type: none"> • Bits #31 to #1 - Reserved (must be set to 0) • Bit #0 - Modulation type: <ul style="list-style-type: none"> – 0 - frequency modulated (such as MSK, FSK2, FSK4, Multitone) – 1 - amplitude modulated (such as ASK2)
12	float_spSampleRate ptypFLOAT_SP	Sample rate of data in data body [Hz]

Word position in frame	Member name Member type	Description
13	float_spSamplesPerSymbol ptypFLOAT_SP	Samples per symbol - number of samples per demodulated symbol. Depends on the used oversampling in the demodulator. Is usually in the range 6 to 20 samples per symbol.
14	uintSampleCount ptypUINT	Sample count: number of samples to follow this header



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Instantaneous Data Body

The Instantaneous Data body contains the actual Instantaneous Data samples. The samples are sent as an array of type `ptypFLOAT_SP`. The data body size is given by: `Sample count` [32-bit words]. In case of frequency modulated signals, the samples represent the instantaneous frequency in Hz and in case of amplitude modulated signals, the samples represent the measured antenna voltage in μV .

8 Decoder Datastreams

8.1 Image Data Format

The Image Data format is used for demodulation result image data. It describes the `ekFRH_DATASTREAM__IMAGEDATA` datastream type.

Image Data Frame Structure

The structure of the Image Datastream is defined in the `rs_gx40x_imagedata_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

The image is split into lines, containing the same number of pixels per line, with pixels of same size (bit representation). Each frame can contain one or several lines of the picture with the last line in the image being specially marked in the frame header. Thus the total number of lines in the image can be deduced only after the last line is transmitted.

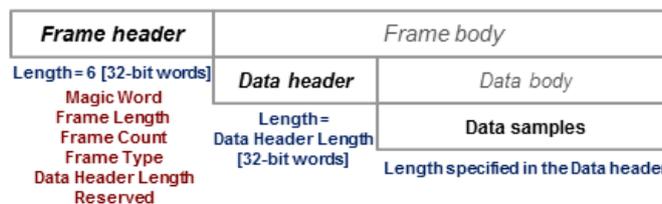


Fig. 8-1: Image Data frame format

NOTICE

Limited use of the frame structure definition: `typIMAGEDATAFRAME`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Image Data Header

The **basic** *Data header* contains a number of fields, that are always sent.

The **extended** *Data header* contains extra information fields sent after the fields of the basic structure.

The length of the *Data header*, as specified by the `uintDataHeaderLength` parameter from the *Frame header*, gives information about which *Data header* type is used, the basic or the extended one.

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The Image *Data header* structure, of type `typIMAGEDATAHEADER`, is described in the following table (*Data header* length = 3).

Table 8-1: Image Data header (typIMAGEDATAHEADER)

Word position in frame	Member name Member type	Description
7	<code>uintStatusword</code> <code>ptypUINT</code>	<p>Status word (bit-coded):</p> <ul style="list-style-type: none"> • Bits #31 to #5 - Reserved (have to be set to 0) • Bit #4 - Data type <ul style="list-style-type: none"> – 0 indicates data type = image – 1 indicates data type = text • Bits #3 to #2 - Pixel representation: <ul style="list-style-type: none"> – 0 indicates 1-bit pixel representation (pixels as binary values: 0 or 1). One 32-bit word carries the information of 32 pixels. – 1 indicates 8-bit pixel representation (luminance information, 256 grayscale values, the lighter the tone, the higher the numeric value). One 32-bit word carries the information of 4 pixels. – 2 indicates 24-bit pixel representation (color information in RGB format, each color being represented on 8 bits, 256 values, the lighter the tone, the higher the numeric value). One 32-bit word carries the information of one pixel: <ul style="list-style-type: none"> bits #31 to #24 – not used, set to 0 bits #23 to #16 – RED level bits #15 to #08 – GREEN level bits #07 to #00 – BLUE level • Bit #1 - Last line indicator - if this flag is set to 1 than this frame contains the last line of the current image • Bit #0 - Image direction - set to 0 when the image is sent from top line to bottom line and 1 otherwise
8	<code>uintPixelsPerLine</code> <code>ptypUINT</code>	Number of Pixels per line for the current image. In case of text data this gives the number of ASCII characters.
9	<code>uintNoOfLines</code> <code>ptypUINT</code>	Number of lines (image lines) sent in this frame. In case of text data, this has to be set to 1.

The Extended Image *Data header* structure, of type `typIMAGEDATAHEADER_EX`, is described in the following table (*Data header* length = 5).

Table 8-2: Extended Image Data header (typIMAGEDATAHEADER_EX) - extra fields only

Word position in frame	Member name Member type	Description
10	bigtimeTimeStamp	64-bit Timestamp [µs] - Absolute time of the first data sample in this frame.
11	ptypBIGTIME	



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Image Data Body

The Image Data body contains the actual Image Data samples, i.e. the pixel information. Each image is defined as a set of lines with each line having a certain number of pixels. One frame can contain the pixel information of one or more lines, as specified by the *Number of lines* parameter in the Data header.

The total number of pixels sent in the current frame is given by: *Number of lines* * *Number of Pixels per line*. If the size of the data type, specified in the Status word, is smaller than ptypUINT, multiple pixels or ASCII characters get packed into a ptypUINT word. Therefore the size of a line depends on the number of pixels and the size of the pixel representation.

The following *LineSize* variable defines the size of a line in [32-bit words] for different pixel representation types:

- 1-bit pixel representation => $LineSize = \text{ceil}(\text{uintPixelsPerLine}/32)$ [32-bit words]
- 8-bit pixel representation => $LineSize = \text{ceil}(\text{uintPixelsPerLine}/4)$ [32-bit words]
- 32-bit pixel representation => $LineSize = \text{uintPixelsPerLine}$ [32-bit words]
- ASCII Character => $LineSize = \text{ceil}(\text{uintPixelsPerLine}/4)$ [32-bit words]

Unused bits in the last 32-bit word of EACH line are set to zero!

The entire data body size is given by: $LineSize * Number\ of\ lines$ [32-bit words].

The following table presents an example of a Data body structure that contains five lines of an image having 10 pixels per line and using the 8-bit pixel representation. For each line a number of $\text{ceil}(10/4) = 3$ [32-bit words] are needed to represent a line of the image. The data body containing five lines has a total of $5 * 3 = 15$ 32-bit words.

	32-bit word				32-bit word				32-bit word			
byte	1	2	3	4	1	2	3	4	1	2	3	4
line1	pixel 1	pixel 2	pixel 3	pixel 4	pixel 5	pixel 6	pixel 7	pixel 8	pixel 9	pixel 10	0	0
line2	pixel 1	pixel 2	pixel 3	pixel 4	pixel 5	pixel 6	pixel 7	pixel 8	pixel 9	pixel 10	0	0
line3	pixel 1	pixel 2	pixel 3	pixel 4	pixel 5	pixel 6	pixel 7	pixel 8	pixel 9	pixel 10	0	0

	32-bit word				32-bit word				32-bit word			
byte	1	2	3	4	1	2	3	4	1	2	3	4
line4	pixel 1	pixel 2	pixel 3	pixel 4	pixel 5	pixel 6	pixel 7	pixel 8	pixel 9	pixel 10	0	0
line5	pixel 1	pixel 2	pixel 3	pixel 4	pixel 5	pixel 6	pixel 7	pixel 8	pixel 9	pixel 10	0	0

8.2 Decoded Text Data Format

The Decoded Text Data format is used for decoder result text data (Bitstream Processing Channel Decoding). It describes the `ekFRH_DATASTREAM__DECODER_TEXT_DATA` datastream type.

Decoded Text Data Frame Structure

The structure of the Decoded Text Datastream is defined in the `rs_gx40x_decoder_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

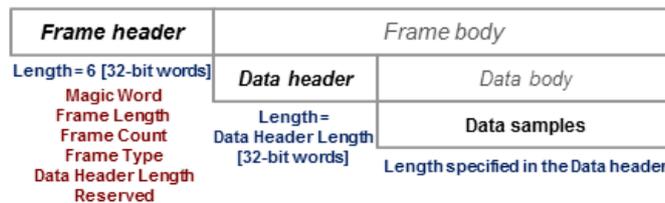


Fig. 8-2: Decoded Text Data frame format

NOTICE

Limited use of the frame structure definition: `typDECODER__DECODED_TXT_FRAME`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Decoded Text Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The Decoded Text *Data header* structure, of type `typDECODER__DECODED_TXT_HEADER`, is described in the following table (*Data header* length = 4).

Table 8-3: Decoded Text Data header (`typDECODER__DECODED_TXT_HEADER`)

Word position in frame	Member name Member type	Description
7 8	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [µs] - Absolute time of the first data sample in this frame from which the results were calculated.
8	unModuleID ptypUINT	Module ID - The system wide unique identifier of the decoder.
9	unCharElement-Count ptypUINT	Number of characters - The number of decoded characters (elements) in this frame.



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Decoded Text Data Body

The Decoded Text Data body contains the decoded text payload which is defined as an array with `unCharElementCount` elements of type `typDECODER__DECODED_CHARELEMENT`.

Table 8-4: Decoded Text data sample format (`typDECODER__DECODED_CHARELEMENT`)

Member name Member type	Description
unSubChannelID ptypUINT	Sub-Channel ID - The ID of the sub-channel to which this decoded character belongs. Sub-Channel Identifiers are used to implement a logical separation of decoded text into channels. Some methods (e.g. Packet Radio) multiplex data into sub-channels. These sub-channels are detected by the decoder, and the decoded data packaged accordingly into sub-channels in the decoded data output stream. Some methods (e.g. PICCOLO) do not provide redundancy that allows for unambiguous interpretation during decoding. When the decoder detects ambiguity, it opens a sub-channel in the decoded data output stream, containing the characters from the alternative symbol interpretation. Similarly, sub-channels are closed when the unambiguity no longer exists
decoded_charDecodedChar typDECODER__DECODED_CHAR	Decoded character. The <code>typDECODER__DECODED_CHAR</code> type is defined as a 32-bit unsigned integer.

8.3 Transmission System Result (TSR) Data Formats

The TSR Data format is used for datastreams containing the results of the digital demodulation and decoder system for different transmission methods. It describes the `ekFRH_DATASTREAM__TRANSMISSION_SYSTEM_RESULT_DATA` datastream type.

TSR Data Frame Structure

The structure of the Transmission System Result Datastream is defined in the `rs_gx40x_transmission_system_result_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

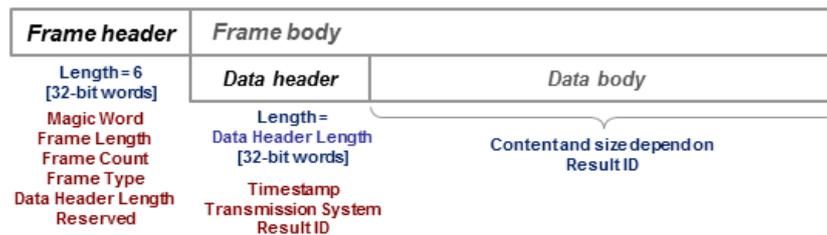


Fig. 8-3: TSR Data frame format

NOTICE

Limited use of the frame structure definition: `typTRANSMISSION_SYSTEM_RESULT_FRAME`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

TSR Data Header

The *Data header* identifies the transmission method and the corresponding format of the data samples.

The TSR *Data header* structure, of type `typTRANSMISSIONSYSTEMDATAHEADER`, is described in the following table (*Data header* length = 7).

Table 8-5: TSR Data header (typTRANSMISSIONSYSTEMDATAHEADER)

Word position in frame	Member name Member type	Description
7 8	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [µs] - Absolute time of the first data sample in this frame from which the results were calculated.
9 to 12	packedcharTransmissionSystem ptypPACKEDCHAR	Transmission System Name - defined as an array of kTRANSMISSION_SYSTEM_STRING_LEN=4 elements of type ptypPACKEDCHAR. Contains up to 16 ASCII (ANSI X3.4) characters that define the Transmission system name of the result data in the frame. Empty fields are filled with 0. Note: A name of exactly 16 characters has no terminating \0.
13	uintStatusword ptypUINT	Status word (bit-coded): <ul style="list-style-type: none"> • Bits #31 to #8 - Reserved (must be set to 0) • Bits #7 to #0 - Result ID - this field identifies the format of the result data (typTRANSMISSIONSYSTEMRESULTDATA) contained in this frame. For each transmission system type (xxx) the corresponding result ID is defined in the header file as constant of the form: kTRANSMISSION_SYSTEM_ID_XXX. The corresponding format of the result data is defined as a structure: typTRANSMISSIONRESULT_XXX. The supported transmission system types are listed in the table below. Each result data format will be described in detail in the next section.



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.



Fig. 8-4: Structures used for the TSR datastreams

Result ID value	Transmission method	Transmission method description	Result data format
0	ASCII	ASCII (American Standard Code for Information Interchange) - 8-bit characters	"Result data format: ASCII" on page 54
1	UNICODE	UNICODE - 16-bit characters	"Result data format: UNICODE" on page 54
2	ATIS	ATIS (Automatic Transmission Identification System) - Maritime identification information	"Result data format: ATIS" on page 54

Result ID value	Transmission method	Transmission method description	Result data format
3	FMS_BOS	FMS-BOS - German abbreviation for FunkMeldeSystem für Behörden und Organisationen mit Sicherheitsanforderungen (radio communications system for authorities and organizations with security concerns (police, fire brigade, customs, ambulances))	"Result data format: FMS-BOS" on page 54
4	ZVEI	ZVEI - system of the German institution Zentralverband Elektrotechnik und ElektronikIndustrie e.V. (Central Association of Electrical and Electronic Industries)	"Result data format: ZVEI" on page 56
5	ZVEI_VDEW	ZVEI-VDEW - system of the German institution Vereinigung Deutscher ElektrizitätsWerke e.V. (Union of German Power Companies)	"Result data format: ZVEI-VDEW" on page 57
6	POCSAG	POCSAG - system of the Post Office Code Standardization Advisory Group (pager system described in CCIR Recommendation 584, Radiopaging Code 1)	"Result data format: POCSAG" on page 58
7	PACKET_RADIO	Packet Radio - Packet switching technology on radio link instead of fixed connection line between stations	"Result data format: PACKET_RADIO" on page 59
8	MPT1327	MPT1327 - system of the Ministry of Postal service and Telecommunications (UK) - trunked radio network standard	"Result data format: MPT1327" on page 62
9	ACARS	ACARS - Aircraft Communications Addressing and Reporting System (Aircraft communication)	"Result data format: ACARS" on page 63
10	PACTOR	PACTOR_II and PACTOR_III - PACKet Teleprinting Over Radio	"Result data format: PACTOR II and III" on page 64
11	CLOVER	CLOVER2 and CLOVER2000 - adaptive modulation system	"Result data format: CLOVER" on page 66
12	F7W	F7W - F: frequency modulation, 7: multichannel digital signal, W: mixed information, according to ITU recommendation - Frequency-Shift-Keying, 2 independent channels	"Result data format: F7W" on page 67
13	MORSE	Morse - Telegraphy system	"Result data format: F7W" on page 67
16	PDU	Generic format according to PDU concept	"Result data format: PDU" on page 69

TSR Data Body

The Data body structure is given by the Result ID value contained in the Status Word of the Data header. Each TRS result type is assigned a Result ID for which a specific TRS data format is defined (for example typTRANSMISSIONRESULT_ASCII).

❖ **Result data format: ASCII***Table 8-6: Data body format for ASCII result datastream (typTRANSMISSIONRESULT_ASCII)*

Word position in Data body	Member name Member type	Description
1	uintDataCount ptypUINT	Datablock size (in units of 32-bit words)
2 ...	packedcharMsg ptypPACKEDCHAR	Datablock containing ASCII characters - defined as an array of size <code>uintDataCount</code> with elements of type <code>ptypPACKEDCHAR</code> . Each element of type <code>ptypPACKEDCHAR</code> contains up to four ASCII characters. The first of the four characters occupies the least significant bits (#0 to #7) of the 32-bit element. The unused space of the last element in the array is padded with NULL characters.

❖ **Result data format: UNICODE***Table 8-7: Data body format for UNICODE result datastream (typTRANSMISSIONRESULT_UNICODE)*

Word position in Data body	Member name Member type	Description
1	uintDataCount ptypUINT	Datablock size (in units of 32-bit words)
2 ...	packedunicodeMsg ptypPACKEDUNICODE	Datablock containing UNICODE characters - defined as an array of size <code>uintDataCount</code> with elements of type <code>ptypPACKEDUNICODE</code> . Each <code>ptypPACKEDUNICODE</code> element contains two UNICODE characters. The first of the two characters occupies the least significant bits (#0 to #15) of the 32-bit element. The unused space of the last element in the array is padded with the NUL character.

❖ **Result data format: ATIS***Table 8-8: Data body format for ATIS result datastream (typTRANSMISSIONRESULT_ATIS)*

Word position in Data body	Member name Member type	Description
1	uintCountryCode ptypUINT	Country Code , Values: 0-999
2 3	packedcharShipID ptypPACKEDCHAR	Ship Identifier - ASCII string of 5 characters. Defined as an array containing <code>kATIS_SHIPID_LEN = 2</code> elements of type <code>ptypPACKEDCHAR</code>

❖ **Result data format: FMS-BOS**

The FMS-BOS decoder evaluates the main message block from which it extracts the sent information. The following message blocks will be ignored by the decoder.

Table 8-9: Data body format for FMS-BOS result datastream (typTRANSMISSIONRESULT_FMS_BOS)

Word position in Data body	Member name Member type	Description
1	uintBOS ptypUINT	BOS ID (4 bits) - Authority/Organization Identifier <ul style="list-style-type: none"> • 1 - Polizei • 2 - Bundesgrenzschutz • 3 - Bundeskriminalamt • 4 - Katastrophenschutz • 5 - Zoll • 6 - Feuerwehr • 7 - Technisches Hilfswerk • 8 - Arbeiter-Samariter-Bund • 9 - Deutsches Rotes Kreuz • 10 - Johanniter-Unfall-Hilfe • 11 - Malteser-Hilfsdienst • 12 - Deutsche Lebensrettungsgesellschaft • 13 - Rettungsdienst • 14 - Zivilschutz • 15 - Fernwirktelegramme
2	uintState ptypUINT	State/country Code (4 bits) <ul style="list-style-type: none"> • 1 - Bund • 2 - Baden-Württemberg • 3 - Bayern I • 4 - Berlin • 5 - Bremen • 6 - Hamburg • 7 - Hessen • 8 - Niedersachsen • 9 - Nordrhein-Westfalen • 10 - Rheinland-Pfalz • 11 - Schleswig-Holstein • 12 - Saarland • 13 - Bayern II • 14 - Mecklenburg-Vorpommern (District 00:49) Sachsen-Anhalt (District 50:99) • 15 - Brandenburg (District 00:49) Thüringen (District 50:99)
3	uintDistrict ptypUINT	District identifier (4 bits) , Values: 0-0xFF
4	packedcharVehicle ptypPACKEDCHAR	Vehicle identifier (4 BCD digits)
5	uintStatus ptypUINT	Call Status (4 bits) . Direction dependent interpretation. For details see table 8-10
6	uintZBV ptypUINT	ZBV Identifier - special purpose fields. <ul style="list-style-type: none"> • Bits #3 and #2 - Special purpose indication 1, 2, 3 and 4 corresponding to values 00, 01, 10 and 11 • Bit #1 - Direction indicator - the value 0 indicates mobile party to base and the value 1 indicates base to mobile party • Bit #0 - ACK indicator - the value 0 indicates system with tonal acknowledge and the value 1 indicates system with digital acknowledge

Table 8-10: FMS-BOS Status identifier (uintStatus)

value	Direction: Base to mobile	Direction: Mobile to base
0	Status query Statusabfrage	Emergency Notruf
1 to 7	BOS-service specific designation Festlegung nach dienstspezifischer BOS-Ver- einbarung	BOS-service specific designation Festlegung nach dienstspezifischer BOS-Ver- einbarung
8	Remote signaling criterion #1 Fernwirkkriterium I	BOS-service specific designation Festlegung nach dienstspezifischer BOS-Ver- einbarung
9	Remote signaling criterion #2 Fernwirkkriterium II	Manual acknowledge / 3rd party login Handquittung/Fremdanmeldung
10	Initiate extended message Vorbereitung Folgetelegramm	Initiate extended message Vorbereitung Folgetelegramm
11	End extended message Ende Folgetelegramm	End extended message Ende Folgetelegramm
12 13	Available for digital pagers Frei für digitale Meldeempfänger	Available for special purpose usage Frei für Sonderanwendungen
14	Available for digital pagers Frei für digitale Meldeempfänger	Automatic acknowledge Automatische Quittung
15	Automatic acknowledge Automatische Quittung	Talk-key Sprechtaste

❖ Result data format: ZVEI

Table 8-11: Data body format for ZVEI result datastream (typTRANSMISSIONRESULT_ZVEI)

Word position in Data body	Member name Member type	Description
1	boolFrameChecksumIsCorrect ptypBOOL	CRC check result , Values: pkTRUE or pkFALSE (pkTRUE means CRC checksum is correct)
2	uintBAK ptypUINT	Mode of operation ID (BAK is the German abbreviation for "Betriebsartenkennung"), Values: 0-15. For details see: table 8-13
3	uintStatus ptypUINT	Status - custom configurable content, Values: 0-15
4	uintEVU ptypUINT	Manufacturer and model identification (EVU is the German abbreviation for Energie Versorgungs Unternehmen = electric utility organization) <ul style="list-style-type: none"> • Bits #15 to #8 - Transmitting device model identification • Bits #7 to #0 - Transmitting device manufacturer identification
5	packedcharCallNumber ptypPACKEDCHAR	Call number - ASCII string

❖ Result data format: ZVEI-VDEW

Table 8-12: Data body format for ZVEI-VDEW result datastream (typTRANSMISSIONRESULT_ZVEI_VDEW)

Word position in Data body	Member name Member type	Description
1	boolFrameChecksumIsCorrect ptypBOOL	CRC check result (pkTRUE means CRC checksum is correct)
2	uintBAK ptypUINT	Mode of operation ID (BAK is the German abbreviation for "Betriebsartenkennung"), Values: 0–15. For details see: table 8-13
3	uintStatus ptypUINT	Status - custom configurable content, Values: 0–15. Only relevant for ZVEI.
4	uintEVU ptypUINT	Manufacturer and model identification (EVU is the German abbreviation for Energie Versorgungs Unternehmen = electric utility organization) <ul style="list-style-type: none"> • Bits #15 to #8 - Transmitting device model identification • Bits #7 to #0 - Transmitting device manufacturer identification
5	uintDataCount ptypUINT	Size of the following call number, in 32-bit data words, Values: 0–1023
6 ...	packedcharCallNumber ptypPACKEDCHAR	Call number - ASCII string (size given by uintDataCount in 32-bit words)

Table 8-13: BAK - Mode of operations for ZVEI and ZVEI-VDEW transmission systems (uintBAK)

value	Interpretation for ZVEI		Interpretation for ZVEI-VDEW	
	German description	English description	German description	English description
0	Frei verfügbar	Freely available	Frei verfügbar	Freely available
1	Ruf zum Fahrzeug ¹	Call to vehicle unit ¹	Ruf zum Mobilstation ¹	Call to mobile unit ¹
2	Ruf zum Leitstelle ¹	Call to base unit ¹	Durchwahl ¹	Direct dial ¹
3	Kennung	Identification	Kennung	Identification
4	Quittung passiv	Acknowledge - passive	Quittung passiv	Acknowledge - passive
5	Reserved		Folgeinformation	Pursuant information
6	Trennruf ¹	Disconnect ¹	Schlussruf ¹	Disconnect ¹
7	Reserved		Rueckruf ¹	Callback ¹
8	Vorrangruf ¹	Priority call ¹	Notruf ¹	Emergency call ¹
9	Statusabfrage ¹	Status query ¹	Statusabfrage/ -antwort ¹	Status query / response ¹
10	Reserved		Lockruf (Positionskontrolle) ¹	Birdcall (position check) ¹
11	Quittung aktiv	Acknowledge - active	Quittung aktiv	Acknowledge - active

value	Interpretation for ZVEI		Interpretation for ZVEI-VDEW	
	German description	English description	German description	English description
12	Reserved		Kurzwahl ¹	Abbreviated-dialing ¹
13	Frei verfügbar	Freely available	Daten	Data
14	Frei verfügbar	Freely available	Reserved	
15	Notruf	Emergency call	Nicht belegt	Unused

Note: ¹ Requires receipt



Result data format: POCSAG

Table 8-14: Data body format for POCSAG result datastream (typTRANSMISSIONRESULT_POCSAG)

Word position in Data body	Member name Member type	Description
1	boolFrameCheckSumIsCorrect ptypBOOL	CRC check result <ul style="list-style-type: none"> pkTRUE means CRC checksum is correct pkFALSE indicates that a non-recoverable CRC was detected, and the data part (packedcharMsg) is incomplete
2	uintBaudrate ptypUINT	Baud rate (can have one of the following values: 512, 1200, 2400 symbols/s)
3	uintFullAddress ptypUINT	Address specifier for the called unit <ul style="list-style-type: none"> Bits #22 to #5 - Transmitting device identification Bits #4 to #2 - Group ID Bits #1 to #0 - this field indicates the content type of the message payload. ITU-R M.584-2 defines two message types: 4-bit numeric only and 7-bit (International Alphabet #5) alpha-numeric <ul style="list-style-type: none"> 00 - indicates that the message uses 4-bit characters which are mapped into a set of (ASCII) characters containing: numeric characters 0... 9, left and right square brackets [and], the hash character # (representing SPARE), character U as urgent indicator, space character, hyphen character - 11 - indicates that the message uses 7-bit characters (ASCII) defined as Alphabet #5 on the CCITT standard 01 or 10 - indicates that the message type is agreed between the transmitting party and the receiving party. The transmitted data is placed in the message payload as a sequence of bits; the bit of least significance of the first element in the data payload is the first bit in the transmission

Word position in Data body	Member name Member type	Description
4	uintDataCount ptypUINT	Size of the following message (in units of 32-bit words), Values: 0-1023
5 ...	packedcharMsg ptypPACKEDCHAR	Message - ASCII string (defined as an array of ptypPACKEDCHAR elements with size uintDataCount). A message is transmitted as a sequence of 32-bit words. Each word carries 20 bits of data, and 12 redundancy bits. When a CRC error is detected in the transmitted message data, the receiver will terminate assembling data. The data is then made available, along with the CRC error indication. In this case the data payload will only contain the message data assembled from the beginning of the message, up to the point at which the CRC error was detected. Each time a new message is started, all the bits of the data payload are initialized to 0. Thus if a message is interrupted (i.e. a CRC error was detected), then all the bits in the un-filled part of the payload contain 0.

❖ Result data format: PACKET_RADIO

Table 8-15: Data body format for PACKET_RADIO result datastream (ptypTRANSMISSIONRESULT_PACKET_RADIO)

Member name Member type	Description
boolFrameChecksumsCorrect ptypBOOL	CRC check result <ul style="list-style-type: none"> • pkTRUE means CRC checksum is correct • pkFALSE indicates that a non-recoverable CRC was detected, and the data part (packedcharMsg) does not contain all the data from the transmission
uintFrameType ptypUINT	Frame type identifier <ul style="list-style-type: none"> • Frame type Information - 0 • Frame type Supervisory - 1 • Frame type Unnumbered - 2
uintPID ptypUINT	Protocol Identifier - indicates the ISO/OSI level 3 protocol used by message, Values: 0-255 <ul style="list-style-type: none"> • AX.25 layer 3 implemented Binary – xx01xxxx or xx10xxxx (where the x positions can have any value) • ISO 8208/CCITT X.25 PLP - 0x01 • Compressed TCP/IP - 0x06 • Uncompressed TCP/IP - 0x07 • Segmentation fragment - 0x08 • TEXNET datagram protocol - 0xC3 • Link Quality protocol - 0xC4 • Appletalk - 0xCA • Appletalk ARP - 0xCB • ARPA Internet Protocol - 0xCC • ARPA Address Resolution - 0xCD • FlexNet - 0xCE NET/ROM - 0xCF • Layer 3 protocol implemented - 0xF0 • Escape character indicating that the next octet contains further (alternative) Level 3 protocol information - 0xFF

Member name		Description	
Member type			
packetradioaddressAddressField typPACKETRADIO_ADDRESS_FIELD		The Address Field identifies both the source of the frame and its destination. In addition, the address field contains the command/response information and facilities Layer 2 repeater operation. The structure <code>typPACKETRADIO_ADDRESS_FIELD</code> that encapsulates the address field contains following elements:	
{	packetradioaddressSource typPACKETRADIO_ADDRESS	Source address	
	packetradioaddressRouter1 typPACKETRADIO_ADDRESS	Address of the first router	
	packetradioaddressRouter2 typPACKETRADIO_ADDRESS	Address of the second router	
	packetradioaddressDestination typPACKETRADIO_ADDRESS	Destination address	
}			
		The structure <code>typPACKETRADIO_ADDRESS</code> that encapsulate the address information contains the following elements:	
{	packedcharCallSign ptypPACKEDCHAR	CallSign - ASCII string (defined as an array of <code>ptypPACKEDCHAR</code> elements with size <code>kPACKETRADIO_CALLSIGN_LEN=2</code>). Not used space is filled with Space characters.	
	uintSSID ptypUINT	SSID - Secondary Station Identifier, Values: 0-15. Used to separate destinations with identical CallSign	
	boolFlag ptypBOOL	Flag - Command/Response/H-Bit. The H-Bit indicates that the Layer 2 of the repeater station has repeated the frame. Values: <code>pkTRUE</code> , <code>pkFALSE</code> .	
}			
packetradiocontrol fieldControlField typPACKETRADIO_CONTROL_FIELD		Control-Field - the union <code>typPACKETRADIO_CONTROL_FIELD</code> that encapsulates the info field contains three structures associated to the three possible frame types: Information, Supervisory and Unnumbered. The elements of these structures are described below:	
{	I	uintNS ptypUINT	Send sequence number , Values: 0-127
		uintNR ptypUINT	Receive sequence number , Values: 0-127
		boolFlag ptypBOOL	Poll/Final bit , Values: <code>pkTRUE</code> , <code>pkFALSE</code>

Member name		Description	
Member type			
	U	uintM ptypUINT	Modifier function , Values: 0-9 <ul style="list-style-type: none"> • 0 UI Unnumbered Information Frame • 1 DM Disconnect Mode • 2 SABM Connect Request • 3 DISC Disconnect request • 4 UA Unnumbered Acknowledge • 5 SABME Connect Request Extended • 6 FRMR Frame Reject • 7 XID Exchange Identifications • 8 TEST Test • 9 MF_UNKNOWN Unknown, function type could not be identified correctly
		boolFlag ptypBOOL	Poll/Final bit , Values: pkTRUE, pkFALSE
	S	uintS ptypUINT	Supervisory function , Values: 0-4 <ul style="list-style-type: none"> • 0 RR Receive Ready • 1 RNR Receive Not Ready • 2 REJ Reject Frame • 3 SREJ Selective Reject • 4 SF_UNKNOWN Unknown, function type could not be identified correctly
		uintNR ptypUINT	Receive sequence number , Values: 0-127
boolFlag ptypBOOL		Poll/Final bit , Values: pkTRUE, pkFALSE	
}			
packetradioinfofield InfoField ptypPACKETRADIO_INFO_FIELD		Info-Field - the structure <code>typPACKETRADIO_INFO_FIELD</code> that encapsulates the info field contains following elements:	
{	uintDataCount ptypUINT	Size of the following message (in units of 32-bit words), Values: 0-256. The value 0 is used to signal the link disconnection action	
	packedcharMsg ptypPACKEDCHAR	Message - ASCII string not necessarily NULL terminated (defined as an array of <code>ptypPACKEDCHAR</code> elements with size <code>uintDataCount</code>)	
}			

❖ Result data format: MPT1327

Table 8-16: Data body format for MPT1327 result datastream (typTRANSMISSIONRESULT_MPT1327)

Member name Member type	Description
uintStatusWord ptypUINT	Status Word - determines the content type of the datablock: <ul style="list-style-type: none"> • Bits #31 to #3 - Reserved; must always be 0 • Bit #2 - set to 1 indicates that the payload data is of type Message-Datablock • Bit #1 - set to 1 indicates that the payload data is of type Connection-Datablock • Bit #0 - set to 1 indicates that the payload data is of type ControlChannel-Block
mpt1327Data typTRANSMISSIONRESULT_MPT1327_DATA	Datablock for the MPT 1327 transmission mode - the union typTRANSMISSIONRESULT_MPT1327_DATA contains three structures associated to control, connection and message data types as presented below:
{ mptControlChannelData typMPT_1327_CONTROL_CHANNEL_BLOCK	Structure typMPT_1327_CONTROL_CHANNEL_BLOCK for the MPT 1327 - Control Channel System Codeword (CCSC). Sent for identification of the base station, usually sent only once.
{ uintSystemID ptypUINT	Unique ID of the MPT 1327 TSC (Trunked System Controller)
uintControlChannelID ptypUINT	Channel number of the control channel, Values: 0-1024
uintControlChannelFrequency_Low ptypUINT	64-bit Frequency of the control channel [Hz] - least significant 32 bits (_Low) followed by most significant 32 bits (_High)
uintControlChannelFrequency_High ptypUINT	
}	
mptConnectionData typMPT_1327_CONNECTION_BLOCK	Structure typMPT_1327_CONNECTION_BLOCK for the MPT 1327 - Every activity in the network is reported through a connection datablock. Address Information for the active connection:
{ uintTransmissionType ptypUINT	The type of the intercepted transmission (see the corresponding definitions at the beginning of the header file): <ul style="list-style-type: none"> • VOICE_TRANSMISSION - 0 • DATA_TRANSMISSION - 1 • SHORT_DATA_MESSAGE - 2 • SYSTEM_OPERATION - 3
packedcharFromRadio ptypPACKEDCHAR	Calling Radio Unit identifier (PREFIX - 7bit prefix and IDENT - 13bit identifier) (array of ptypPACKEDCHAR elements with size kMPT1327_RADIO_LEN=2)
packedcharToRadio ptypPACKEDCHAR	Called Radio Unit identifier (PREFIX and IDENT) (array of ptypPACKEDCHAR elements with size kMPT1327_RADIO_LEN)=2

Member name		Description
Member type		
	uintChannelNumber ptypUINT	The channel allocated for VOICE_TRANSMISSION or DATA_TRANSMISSION, Values: 0-1024
	uintChannelFrequency_Low ptypUINT	64-bit Frequency of the allocated channel [Hz] - least significant 32 bits (_Low) followed by most significant 32 bits (_High)
	uintChannelFrequency_High ptypUINT	
	uintAcceptedFrames ptypUINT	Number of error free received frames in percent, Values: 0-100
}		
	mptMessageData typMPT_1327_MESSAGE_BLOCK	Structure typMPT_1327_MESSAGE_BLOCK for the MPT 1327 - Message block
{	uintDataCount ptypUINT	Length of the following message (in units of 32-bit words)
	packedcharMessage ptypPACKEDCHAR	Message (array of ptypPACKEDCHAR elements with size uintDataCount)
}		
}		

❖ **Result data format: ACARS**

Table 8-17: Data body format for ACARS result datastream (typTRANSMISSIONRESULT_ACARS)

Word position in Data body	Member name Member type	Description
1	boolCRCresult ptypBOOL	Result of the CRC check <ul style="list-style-type: none"> pkTRUE means CRC checksum is correct pkFALSE indicates that a non-recoverable CRC was detected, and the data part (packedcharMsg) might be corrupted
2	packedcharCategoryOfOperation ptypPACKEDCHAR	Category of operation (ASCII string) - The Mode characters are divided into two basic categories which refer to Category A or Category B network operation.
3	packedcharAircraftId ptypPACKEDCHAR	Aircraft ID (ASCII string, 7 characters null terminated) - identifies the aircraft with which the ground processor is communicating, broadcast is identified by: ALLCALL (defined as an array of ptypPACKEDCHAR elements with size 2)
4	packedcharLinkType ptypPACKEDCHAR	Link type: uplink or downlink (ASCII string: DOWN, UP)

Word position in Data body	Member name Member type	Description
5	packedcharTechnical Acknowledgment ptypPACKEDCHAR	Technical ACK (ASCII string). Positive ACK in DL is identified by any of the following characters: a to z, A to Z. Positive ACK in UL is identified by any of the following characters: 0 to 9. Negative ACK is identified by the string <code>NAK</code>
6	packedcharLabel ptypPACKEDCHAR	Label code (ASCII string) defines the data packet usage - the list of assigned labels is defined in ARINC Specification 620
7	packedcharLinkBlockId ptypPACKEDCHAR	Data packet Identifier (UBI or DBI). A retransmission is identified by the same LinkBlockID
8	boolMsgIsComplete ptypBOOL	Message is complete if boolMsgIsComplete is pkTRUE
9	uintDataCount ptypUINT	Length of the following message (in units of 32-bit words), Values: 0-55. A NACK message has the message length = 0
10 ...	packedcharMsg ptypPACKEDCHAR	Message of the ACARS datagram (array of ptypPACKEDCHAR elements with size uintDataCount)

❖ Result data format: PACTOR II and III

Table 8-18: Data body format for PACTOR result datastream (typTRANSMISSIONRESULT_PACTOR)

field offset	Member name Member type	Description
0	uintPactorVariant ptypUINT	Pactor Variant Values: 0 - unknown, 2 - PactorII, 3 - PactorIII In case the pactor variant is unknown, the fields at offset 1-6,12-21 will carry the default value 0. The field at offset 9 will carry the default value 0.0. The fields at offsets 7, 8, 10, 11 will still be valid. The field at offset 21 will be the last field in the data body.
1	uintTypeOfBurst ptypUINT	Type of Burst Values: 0 - unknown, 1 - CS (control signal), 2 - DATA (contains user data), 3 - IDLE (sent in chat mode if no data available), 4 - CALLSIGN (signaling information) The CS burst is used in case of ARQ transmissions to confirm the receipt of a burst back to the sender
2	uintSpeedLevel ptypUINT	Speed Level Values: (0 - n/a), 1-6. The values 5 and 6 are valid only for PactorII
3	uintPathMode ptypUINT	Path Mode Values: 0 - n/a, 1 - unknown ² , 2 - normal path , 3 - long path (stations > 20000km apart)
4	uintTransmissionMode ptypUINT	Transmission Mode Values: 0 - n/a, 1 - unknown ² , 2 - ARQ transmission, 3 - unproto (non-ARQ) transmission

Transmission System Result (TSR) Data Formats

field offset	Member name Member type	Description
5	uintIsDataMode ptypUINT	indicates whether Data Mode is used for this packet Values: 0 - n/a, 1 - unknown, 2 - yes (data mode needs longer bursts than normal mode), 3 - no (normal mode is used for this burst) Unknown, is used if uintTypeOfBurst indicates a CS burst
6	uintIsChatMode ptypUINT	indicates whether Chat Mode is being used Values: 0 - n/a, 1 - unknown, 2 - yes, 3 - no Unknown, is used if uintTypeOfBurst indicates a CS burst Note: for future use, currently always value 1
7	float_spBurstLength ptypFLOAT_SP	detected Burst Length in [ms]
8	uintCenterFrequency ptypUINT	detected RF Center Frequency of emission in [Hz]
9	float_spSymbolRate ptypFLOAT_SP	detected Symbol Rate of emission in [Bd]
10	float_spSignalLevel ptypFLOAT_SP	detected Signal Level of burst in [dBμV]
11	float_spSNR ptypFLOAT_SP	detected Signal to Noise Ratio in [dB]
12	uintSideband ptypUINT	detected Sideband Values: 0 - unknown, 1 - upper sideband, 2 - lower sideband
13	uintPacketNumber ptypUINT	a 2-bit Packet Number , Values: 0–3
14	uintDataType ptypUINT	User Data Encoding , Values: 0–7 (see the definition of the kTRS_FACTOR_DATATYPE_xxx constants in the header file). <ul style="list-style-type: none"> • 0 - ASCII-8-bit • 1 - Huffman (normal) • 2 - Huffman (swapped, upper case) • 3 - reserved • 4 - PMC German (normal) • 5 - PMC German (swapped) • 6 - PMC English (normal) • 7 - PMC English (swapped) PMC stands for Pseudo-Markov Compression (proprietary compression method)
15	uintDataMode Suggestion ptypUINT	Data Mode Suggestion Values: 0 - off, 1 - on
16	uintSLchangeoverReq ptypUINT	Speed Level Changeover Request Values: 0 - request off, 1 - request on
17	uintIsQRTPacket ptypUINT	flag describing whether this packet is a QRT packet , (QRT indicates the request to stop data transmission) Values: 0 - no, 1 - yes

field offset	Member name Member type	Description
18	uintCRC ptypUINT	CRC 16-bit checksum, Values: 0-65535
19	uintDecodingQuality ptypUINT	Quality of decoded bytes that are delivered with this result frame, Values: (0 - n/a) 1-15, where 15 denotes best quality
20	uintContentType ptypUINT	Content Type Values: 0 - unknown (so the current visualization is to be maintained), 1 - text data, 2 - binary data Note: for future use, currently always value 0
21	uintDataByteCount ptypUINT	Number of Decoded Bytes in [bytes]. Represents the amount of user data extracted from the pactor message belonging to this result frame; the user data is delivered from offset 22 of the data body onwards if <code>uintDataByteCount > 0</code> Values: 0-kTRS_PACTOR_DATABYTECNT_MAX
22 ...	packedcharData ptypPACKEDCHAR	User Data extracted from the pactor packet (array of <code>ptypPACKEDCHAR</code> elements with <code>size = ((uintDataByteCount-1)/4 + 1)</code>)

Note: ²Unknown is used in the very first burst after start of processing, since the path mode can only be detected after receipt of two bursts



Result data format: CLOVER

Table 8-19: Data body format for CLOVER result datastream (typTRANSMISSIONRESULT_CLOVER)

field offset	Member name Member type	Description
0	uintTypeOfBlock ptypUINT	Type of transmitted data Values: 0 - User Data , 1 - Control information
1	uintCloverHeader ptypUINT	Header information from the clover burst
2	uintSpeedLevel ptypUINT	Speed Level Values: (0 - n/a), 1-6. See definition of constants <code>kTRS_SL_XXX</code> . The data rate is increased by th choice of higher modulation <ul style="list-style-type: none"> • 1: 2DPSK2A • 2: PSK2A • 3: PSK4A • 4: PSK8A • 5: PSK8A/ASK26 • 6: PSK16A/ASK4
3	uintRSCodeEfficiency ptypUINT	Efficiency of the used Reed-Solomon coding <ul style="list-style-type: none"> • 0 - unknown (checksum length is not recognized) • 1 - robust (Data block contains 60% data and 40% checksum) • 2 - normal (Data block contains 75% data and 25% checksum) • 3 - fast (Data block contains 90% data and 10% checksum) • 4 - off (Data block is sent without any checksum)

field offset	Member name Member type	Description
4	uintCloverDataBlock Size ptypUINT	Clover data block size in bytes - Length of the current data block (data block = data header+data+checksum). The size can take one of the following values: [17, 51, 85, 255]
5	uintPacketNumber ptypUINT	Packet Number Note: for future use, currently always value 0
6	uintSideband ptypUINT	detected Sideband Values: 0 - unknown, 1 - upper sideband, 2 - lower sideband
7	uintCenterFrequency ptypUINT	detected RF Center Frequency of emission in [Hz]
8	uintContentType ptypUINT	Content type Values: 0 - n/a, 1 - text data, 2 - binary data Note: for future use, currently always value 1
9	uintDataByteCount ptypUINT	Number of Decoded Bytes in [bytes]. Represents the amount of user data extracted from the message belonging to this result frame; the user data is delivered from offset 10 of the data body onwards if <code>uintDataByteCount > 0</code> Values: 0-kTRS_CLOVER_DATABYTECNT_MAX
10 ...	packedcharData ptypPACKEDCHAR	User Data extracted from the clover packet (array of <code>ptypPACKEDCHAR</code> elements with <code>size = ((uintDataByteCount-1)/4 + 1)</code>)



Result data format: F7W

Table 8-20: Data body format for F7W result datastream (typTRANSMISSIONRESULT_F7W)

Member name Member type	Description
float_spF7WBaudRate ptypFLOAT_SP	Symbol rates for the two logical Bitstreams
uintNumberOfOutputChannels ptypUINT	Number of channels provided in this result frame
transmissionresult_f7w_output_channels typTRANSMISSIONRESULT_OUTPUT_CHANNEL	Channel result data , (defined as an array of <code>typTRANSMISSIONRESULT_OUTPUT_CHANNEL</code> elements with size <code>uintNumberOfOutputChannels</code>). The elements of the structure <code>typTRANSMISSIONRESULT_OUTPUT_CHANNEL</code> are given below:
{	
uintChannel Number ptypUINT	Channel number to identify one message (e.g. for 0..1 the sender generates two independent messages)
uintSubChannel Number ptypUINT	Subchannels are used to present various output formats of the message (e.g. 0..1 -> 2 subchannels per channel)

Member name	Description
Member type	
packedcharSubChannelName ptypPACKEDCHAR	Subchannel name , (defined as an array of ptypPACKEDCHAR elements with size kTRS_OUTPUTCHANNEL_STR_LEN = 10)
uintPayloadType ptypUINT	Payload type of the output channel <ul style="list-style-type: none"> 0 - RAW - in this case the Payload is of type ptypUINT 1 - ASCII - in this case the Payload is of type ptypPACKEDCHAR (4 ASCII characters get packed into one 32-bit word) 2 - MORSE in this case the Payload is of type ptypPACKEDCHAR (one Morse-element (dash, dot, pause, etc.) is packed into one 32-bit word, see table below)
uintElementSize ptypUINT	Size of result element in [bits] (8 bits for ASCII, 32 bits for MORSE, any for RAW)
uintElementCount ptypUINT	Number of result elements of size specified by uintElementSize
uintPayloadSize ptypUINT	Length of the payload uintPackedChannelPayload in units of 32-bit words, bits not used are set to 0
uintChannelPayload ptypUINT	Payload , (defined as an array of ptypUINT elements with size uintPayloadSize)
}	

Table 8-21: Mapping TRS payload value to Morse data

Morse data	Binary value	Hex value	Constant defined in header file
DOT tone	00000000	0x0	KTRS_OUTPUTCHANNEL_MORSE__DOT
DASH tone	00000001	0x1	KTRS_OUTPUTCHANNEL_MORSE__DASH
inter-tone gap	00000010	0x2	KTRS_OUTPUTCHANNEL_MORSE__SHORT_PAUSE
inter-character gap	00000011	0x3	KTRS_OUTPUTCHANNEL_MORSE__MEDIUM_PAUSE
inter-word gap	00000100	0x4	KTRS_OUTPUTCHANNEL_MORSE__LONG_PAUSE
new line	00000101	0x5	KTRS_OUTPUTCHANNEL_MORSE__NEW_LINE
Error messages			
tone too short	10000000	0x40	KTRS_OUTPUTCHANNEL_MORSE__DOT_TOO_SHORT
tone too long	10000001	0x41	KTRS_OUTPUTCHANNEL_MORSE__DASH_TOO_LONG
gap too short	10000010	0x42	KTRS_OUTPUTCHANNEL_MORSE__PAUSE_TOO_SHORT

❖ **Result data format: PDU**

Table 8-22: Data body format for generic PDU result datastream format (typTRANSMISSIONRESULT_PDU)

Member name Member type	Description
uintCOMMSYSTYPE ptypUINT	Unique Communication System ID of the system which produced this result (see also comm_sys.h)
uintPDU_ID ptypUINT	Unique PDU ID , identifying the PDU within this frame
uintNumberOfPDU_Members ptypUINT	Number of PDU members
transmissionresult_pdu_members typTRANSMISSIONRESULT_PDU_MEMBER	PDU member (s) , (defined as an array of typTRANSMISSIONRESULT_PDU_MEMBER elements of length uintNumberOfPDU_Members). The elements of the structure typTRANSMISSIONRESULT_PDU_MEMBER are given below:
{ uintPDU_Member_ElementSize ptypUINT	Element size = s in [bits] - size of the elements in the PDU member payload
uintPDU_Member_ElementCount ptypUINT	Number of elements = n in the PDU member payload
uintPDU_Member_SizeInWords ptypUINT	Payload size of the PDU member uintPDU_Member_SizeInWords = ceil((uintPDU_Member_ElementSize * uintPDU_Member_ElementCount)/(8*pdemSIZEOF(ptypUINT)))
uintPDU_Member ptypUINT	Payload - Sequence of n elements of size s (packed in a container defined as an array of ptypUINT elements of length uintPDU_Member_SizeInWord).
}	

In order to interpret the parameters of the PDU format, an application extension in form of a .dll file is needed along the main application (the file is found in the installation directory of the application: PDU_Converter.dll)

9 Detector Datastreams

9.1 Emission List Data Format

The Emission List Data format describes the `ekFRH_DATASTREAM__EMISSION_LIST_DATA` datastream type.

Emission List Data Frame Structure

The structure of the Emission List Datastream is defined in the `rs_gx40x_emissions_list_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

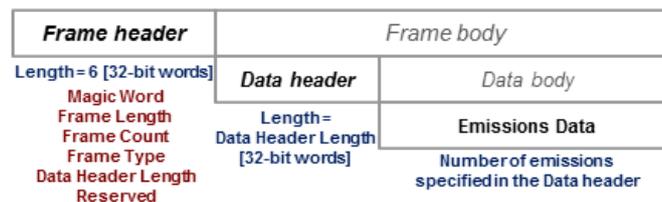


Fig. 9-1: Emission List Data frame format

NOTICE

Limited use of the frame structure definition: `typEMISSION_LIST_FRAME`

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Emission List Data header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The Emission List *Data header* structure, of type `typEMISSION_LIST_HEADER`, is described in the following table (*Data header* length = 5).

Table 9-1: Emission List Data header (*typEMISSION_LIST_HEADER*)

Word position in frame	Member name Member type	Description
7	uintEmissionCount ptypUINT	Emission count: number of emissions in this frame
8	uintStatusword ptypUINT	Status word (bit-coded): <ul style="list-style-type: none"> • Bit #31 - Status of (this) emission tracking table entry containing a list of 0 or more records, each pertaining to an emission. 1 indicates that there are no emissions listed in this emission tracking table entry. • Bits #30 to #4 - Reserved (must be set to 0) • Bits #3 to #0 - Tracking table entry ID. A tracking table entry contains a list of 0 or more records. Each record pertains to an emission.
9	uintCenterFrequency_Low ptypUINT	64-bit Center Frequency of the signal band analyzed, and from which the detected emissions come [Hz] - least significant 32 bits (uintCenterFrequency_Low) followed by most significant 32 bits (uintCenterFrequency_High)
10	uintCenterFrequency_High ptypUINT	
11	uintBandwidth ptypUINT	Bandwidth [Hz] of the analyzed signal band



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Emission List Data body

The Emission List Data body contains the actual Emission Data, defined as a structure of *typEMISSION_DATA* elements. The number of emissions contained in this structure was given in the Data header element: **Emission count**.

Table 9-2: Emission List Data body element (*typEMISSION_DATA*)

Word offset	Member name Member type	Description
0	uintEmissionID ptypUINT	Emission identifier
1 2	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [μ s] - Absolute time of the signal spectrum in which this emission was detected
3	uintCenterFrequency_Low ptypUINT	64-bit Center Frequency of this emission [Hz] - least significant 32 bits (uintCenterFrequency_Low) followed by most significant 32 bits (uintCenterFrequency_High)
4	uintCenterFrequency_High ptypUINT	
5	uintBandwidth ptypUINT	Bandwidth [Hz] of this emission

Word offset	Member name Member type	Description
6	intLevel ptypUINT	Detected Signal level (in dBm/Hz) of this emission
7	signal_statiSignalStatus typSIGNAL_STATI	Current Status of this emission. Following status are possible: <ul style="list-style-type: none"> • ekSIGNAL_STATI_ACTIVE • ekSIGNAL_STATI_INACTIVE • ekSIGNAL_STATI_DECAYED • ekSIGNAL_STATI_UNKNOWN

9.2 Spectral Detector List Data Format

The Spectral Detector List Data format describes the `ekFRH_DATASTREAM__SPECTRALDETECTOR_DATA` datastream type.

Spectral Detector List Data Frame Structure

The structure of the Spectral Detector List Datastream is defined in the `rs_gx40x_spectraldetector_list_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file: `rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

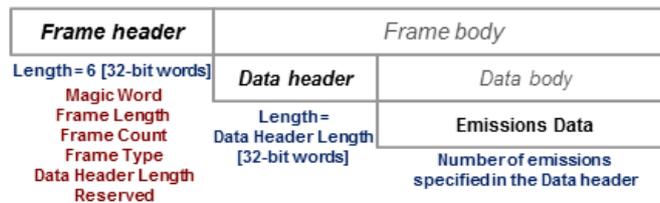


Fig. 9-2: Spectral Detector List Data frame format

Spectral Detector List Data header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The Spectral Detector List *Data header* structure, of type `typSPECTRALDETECTOR_LIST_HEADER`, is described in the following table (*Data header* length = 9).

Table 9-3: Spectral Detector List Data header (*typSPECTRALDETECTOR_LIST_HEADER*)

Word position in frame	Member name Member type	Description
7	uintEmissionCount ptypUINT	Emission count: number of emissions in this frame
8	uintStatusword ptypUINT	Status word (reserved)
9	uintCenterFrequency_Low ptypUINT	64-bit Center Frequency of the signal band analyzed, and from which the detected emissions come [Hz] - least significant 32 bits (<code>uintCenterFrequency_Low</code>) followed by most significant 32 bits (<code>uintCenterFrequency_High</code>)
10	uintCenterFrequency_High ptypUINT	
11	uintBandwidth_Low ptypUINT	64-bit Bandwidth [Hz] of the analyzed band - least significant 32 bits (<code>uintBandwidth_Low</code>) followed by most significant 32 bits (<code>uintBandwidth_High</code>)
12	uintBandwidth_High ptypUINT	
13 14	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [μ s] - Absolute time when the emissions were detected
15	uintAllCategoryCount ptypUINT	AllCategoryCount - the number of categories in the results (user defined categories and generic categories)



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Spectral Detector List Data body

The Spectral Detector List Data body contains the actual Spectral Detector Data, defined as a structure of `typSPECTRALDETECTOR_DATA` elements. The number of emissions contained in this structure was given in the Data header element: **Emission count**.

Table 9-4: Spectral Detector List Data body element (*typSPECTRALDETECTOR_DATA*)

Word offset	Member name Member type	Description
1	uintCenterFrequency_Low ptypUINT	64-bit Center Frequency of this emission [Hz] - least significant 32 bits (<code>uintCenterFrequency_Low</code>) followed by most significant 32 bits (<code>uintCenterFrequency_High</code>)
2	uintCenterFrequency_High ptypUINT	
3	uintBandwidth ptypUINT	Bandwidth [Hz] of this emission

Word offset	Member name Member type	Description
4	intLevel ptypUINT	Detected Signal level (in dBm/Hz) of this emission
5 ...	uintResults ptypUINT	Results (bit-coded) - the number of results is given by the uintAllCategoryCount from the Data header. <ul style="list-style-type: none"> • Bits #31 to #16 - Category ID, ids above 0x8000 reserved for generic categories. The values for the generic categories are defined in the header file. • Bits #15 to #8 - Flags for result. Bit #08 marks the entry as winner, only possible for user categories • Bits #7 to #0 - Confidence for the category, values: [0,100].

9.3 Burst Emission List Data Format

The Burst Emission List Data format describes the `ekFRH_DATASTREAM__BURST_EMISSIONS_LIST` datastream type.

Burst Emission List Data Frame Structure

The structure of the Burst Emission List Datastream is defined in the `rs_gx40x_burst_emissions_list_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file: `rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

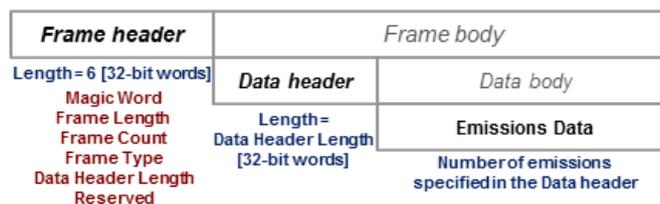


Fig. 9-3: Burst Emission List Data frame format

NOTICE**Limited use of the frame structure definition: typBURST_EMISSION_LIST_DATAFRAME**

The structure definition for the entire frame format, defined in the corresponding header file, is used to illustrate the structure of the data frame and is also suitable for data generation. Do not use this structure for data parsing because new frame versions can use extended header types which might lead to wrong addressing in the frames. Use the **Data Header Length** information from the global *Frame Header* to correctly access the data samples after the *Data Header*.

Burst Emission List Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The Burst Emission List *Data header* structure, of type `typBURST_EMISSION_LIST_DATAHEADER`, is described in the following table (*Data header* length = 9).

Table 9-5: Burst Emission List Data header (typBURST_EMISSION_LIST_DATAHEADER)

Word position in frame	Member name Member type	Description
7 8	uintSnapshotCenterFrequency_Low uintSnapshotCenterFrequency_High ptypUINT	64-bit Snapshot Center Frequency [Hz] - least significant 32 bits (uintCenterFrequency_Low) followed by most significant 32 bits (uintCenterFrequency_High)
9	uintSnapshotBandwidth ptypUINT	Snapshot Bandwidth [Hz]
10 11	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [μ s] - Absolute time of the first snapshot sample
12	uintSnapshotLength ptypUINT	Snapshot duration in ms, of the signal segment in which this emission list was intercepted
13	uintStatusword ptypUINT	Status word (bit-coded): <ul style="list-style-type: none"> • Bits #31 to #1 - Reserved (must be set to 0) • Bit #0 - Snapshot end flag <ul style="list-style-type: none"> – 1 indicates the end of emission of the current signal segment – 0 indicates that emission interception has not yet concluded for the current signal segment
14	uintDatasetCount ptypUINT	Dataset count : Number of emission entries in this frame
15	uintDatasetLength ptypUINT	Length , in 32-bit words of each emission entry in this frame



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Burst Emission List Data Body

The Burst Emission List Data body contains the actual Emission Data, defined as a structure of `typBURST_EMISSION_DATA_SET` elements. The number of emissions contained in this structure was given in the Data header element: **Dataset count**.

Table 9-6: Burst Emission List Data body element (typBURST_EMISSION_DATA_SET)

Word offset	Member name Member type	Description
0	uintEmissionID ptypUINT	Emission identifier
1	uintStartTimeOffset ptypUINT	Start time of this emission relative to first snapshot sample (<code>bigTimeSnapshotStartTime</code>) [μ s]
2	uintDuration ptypUINT	Duration of this emission [μ s]
3	intCenterFrequencyOffset_High ptypINT	Frequency offset [Hz] of this emission, relative to the snapshot center frequency
4	uintBandwidth ptypUINT	Bandwidth [Hz] of this emission
5	float_spMagnitude ptypFLOAT_SP	Average power measured for this emission [dBm]
6	eModulationType typEMISSION_MOD_TYPE	Determined Modulation type . Following modulations are possible: <ul style="list-style-type: none"> • <code>ekEMISSION_MODULATION_TYPE_UNKNOWN</code> • <code>ekEMISSION_MODULATION_TYPE_FSK2</code> • <code>ekEMISSION_MODULATION_TYPE_AM</code> • <code>ekEMISSION_MODULATION_TYPE_PSK2A</code> • <code>ekEMISSION_MODULATION_TYPE_PSK2B</code> • <code>ekEMISSION_MODULATION_TYPE_PSK4A</code> • <code>ekEMISSION_MODULATION_TYPE_PSK4B</code> • <code>ekEMISSION_MODULATION_TYPE_PSK8A</code>
7	float_spModulationTypeParameter1 ptypFLOAT_SP	Modulation specific parameter-1 (determined). The symbol rate [symbols / s] for digital emissions; else meaningless.
8	float_spModulationTypeParameter2 ptypFLOAT_SP	Modulation specific parameter-2 (determined). The frequency shift [Hz] for FSK emissions; else meaningless.

10 Statistics Datastreams

10.1 Histogram Data Format

The Histogram Data format describes the `ekFRH_DATASTREAM__HISTOGRAM_DATA` datastream type.

Histogram Data Frame Structure

The structure of the Histogram Datastream is defined in the `rs_gx40x_histogram_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

`rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

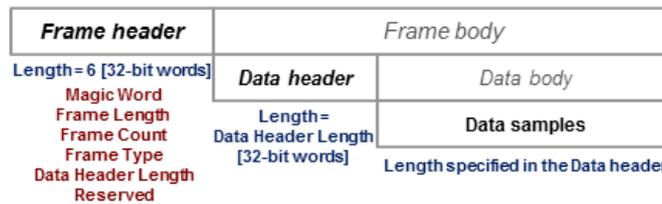


Fig. 10-1: Histogram Data frame format

Histogram Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The Histogram *Data header* structure, of type `typHISTOGRAM_HEADER`, is described in the following table (*Data header* length = 8).

Table 10-1: Histogram Data header (typHISTOGRAM_HEADER)

Word position in frame	Member name Member type	Description
7 8	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [µs] - Absolute time of the first sample of the data from which the histogram data was calculated
9	uintStatusword ptypUINT	<p>Status word (bit-coded):</p> <ul style="list-style-type: none"> • Bits #31 to #5 - Reserved • Bits #4 to #0 - indicates the Histogram data type. Possible values are given by the eHISTOGRAM_DATA enumeration: <ul style="list-style-type: none"> - ekHISTOGRAM_DATA_DURATION = 0x1 - ekHISTOGRAM_DATA_FREQUENCY = 0x2 - ekHISTOGRAM_DATA_BANDWIDTH = 0x3 - ekHISTOGRAM_DATA_LEVEL = 0x4 - ekHISTOGRAM_DATA_SYMBOLRATE = 0x5 - ekHISTOGRAM_DATA_SHIFT = 0x6 - ekHISTOGRAM_DATA_MODULATION_TYPE = 0x7 - ekHISTOGRAM_DATA_TIMING = 0x8 - ekHISTOGRAM_DATA_ENCHANCED_TIMING = 0x9
10 to 13	histBorders typHistogramBorders	<p>Lower and Upper Boundaries of the Histogram define the Histogram range on the x-axis (the intermediate values of the Histogram bins are calculated from the number of bins available). This parameter is defined as a union that contains two structure elements for 64-bit values and float values, respectively (which element is used in this frame is determined by the Statusword - Histogram data type field)</p> <ul style="list-style-type: none"> • stru_uint64 - element used for Histogram data types: 1, 2 and 3. • stru_float_sp - element used for Histogram data types: 4, 5, 6, 8 and 9 <p>Note: In case of Histogram data type number 7 (Modulation Type) there are no Boundaries definitions because the Histogram consists of a predefined number of bins representing the modulation types given by the enumeration typEMISSION_MOD_TYPE (see definition from <i>Burst Emission List Data format</i> table 9-6). For this case, the information contained in the four words histBorders element is not relevant and should be skipped.</p>
	{ stru_uint64	<p>64-bit format for Lower and Upper Histogram Boundaries</p> <p>Note: for C++ implementation the 64-bit format start and end values are provided using the member functions: getStartValue() and getEndValue()</p>
	{ uintStart_Low ptypUINT	64-bit - Start value for histogram bins: least significant 32 bits (uintStart_Low) followed by most significant 32 bits (uintStart_High)
	uintStart_High ptypUINT	
	uintEnd_Low ptypUINT	64-bit - End value for histogram bins: least significant 32 bits (uintEnd_Low) followed by most significant 32 bits (uintEnd_High)
	uintEnd_High ptypUINT }	

Word position in frame	Member name Member type	Description
	stru_float_sp	<i>Floating point single precision format for Lower and Upper Histogram Boundaries.</i> This information is carried in the first two words of the four words <code>histBorders</code> element.
	{ fStart ptypFLOAT_SP	Floating point Start value for histogram bins
	fEnd ptypFLOAT_SP }	Floating point End value for histogram bins
	}	
14	uintBinCount ptypUINT	Number of histogram bins to follow this header Note: In case of Histogram data type number 7 (Modulation Type) the number of bins equals the size of the enumeration <code>typEMISSION_MOD_TYPE</code> .



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Histogram Data Body

The Histogram Data body contains the actual Histogram Bins Data. The number of histogram bins to be read in the Data body was given by the Data header `typHISTOGRAM_HEADER.uintBinCount` parameter. Each histogram bin value is represented on a 32-bit word of type `ptypUINT`. The position on the x-axis of the histogram bins can be calculated from the Lower (L) and Upper (U) Boundaries and the total number of bins that as specified in the Data header. The bin width is given by: $(U-L) / \text{uintBinCount}$.

10.2 Hop Density Waterfall Data Format

The Hop Density Waterfall Data format describes the `ekFRH_DATASTREAM__HOP_DENSITY_WATERFALL_DATA` datastream type.

Hop Density Waterfall Data Frame Structure

The structure of the Hop Density Waterfall Datastream is defined in the `rs_gx40x_hop_density_waterfall_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "[Global Frame header](#)" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file:

rs_gx40x_global_frame_types_if_defs.h.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

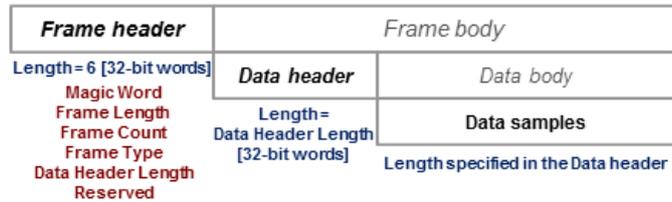


Fig. 10-2: Hop Density Waterfall Data frame format

Hop Density Waterfall Data Header

The *Data header* describes the datastream payload (such as number of Data samples contained in this frame), and contains common parameters of the Data samples.

The Hop Density Waterfall *Data header* structure, of type `typHDW_HEADER`, is described in the following table (*Data header* length = 10).

Table 10-2: Hop Density Waterfall Data header (`typHDW_HEADER`)

Word position in frame	Member name Member type	Description
7 8	bigtimeTimeStamp ptypBIGTIME	64-bit Timestamp [μs] - Absolute time of the first sample of the data from which the hop density waterfall data was calculated
9	uintStatusword ptypUINT	Status word: reserved (currently unused, has value 0).
10 11	uintCenterFrequency_Low uintCenterFrequency_High ptypUINT	64-bit Center Frequency of the of hop density waterfall [Hz] - least significant 32 bits (<code>uintCenterFrequency_Low</code>) followed by most significant 32 bits (<code>uintCenterFrequency_High</code>)
12	uintBandwidth ptypUINT	Bandwidth [Hz] of the hop density waterfall
13	uintSnapshotDuration ptypUINT	Duration of snapshot represented by the bins within this frame
14	uintNumTimeBins ptypUINT	Number of time bins in each waterfall column
15	uintNumFrequencyBins ptypUINT	Number of frequency bins in each waterfall row
16	uintNumIndexValuePairs ptypUINT	Number of pairs - each pair consists of an index (of a vector structure) and a (non-zero) value



The values contained in the data header fields represent the status at the beginning of the frame. A modification happening during the transmission of a frame will only be noted in the data header of the next frame.

Hop Density Waterfall Data Body

The Hop Density Waterfall Data body contains the actual Hop Density Waterfall Data - defined as an array consisting of `uintNumIndexValuePairs` elements of type `typHDW_IndexValuePair`.

Each element consists of an (index, value) pair as described in the table below:

Table 10-3: Hop Density Waterfall Data structure (`typHDW_IndexValuePair`)

Member name Member type	Description
<code>uintIndex</code> <code>ptypUINT</code>	Index within the vectorized matrix where a non zero value (<code>float_spValue</code>) is located
<code>float_spValue</code> <code>ptypFLOAT_SP</code>	The actual value at <code>uintIndex</code> position

How to use the Hop Density Waterfall Data:

Create a zero initialized vector `v` with size `=uintNumTimeBins * uintNumFrequencyBins`.

Set the vector elements using the Index-Value pairs in the Data body. The `uintIndex` represents the location of the `float_spValue` inside the vector `v`.

The Hop Density Waterfall is then represented as a matrix by reordering the vector `v` into a matrix with `uintNumTimeBins` rows and `uintNumFrequencyBins` columns.

11 PDW and IQDW Datastreams

11.1 Pulse Descriptor Words (PDW) Datastream

The Pulse Descriptor Words Data format describes the `ekFRH_DATASTREAM__PULSE_DESCRIPTION_WORD_DATA` datastream type.

PDW Data Frame Structure

The structure of the PDW Datastream is defined in the `rs_tpa_pdw_header_if_defs.h` header file.

The Data Frame consists of the global *Frame header* of type `typFRH_FRAMEHEADER`, as described in "Global Frame header" on page 5, followed by the datastream-specific *Frame body*.

The corresponding "Frame Type" value from the *Frame header* for this datastream type can be found in the global frame types header file: `rs_gx40x_global_frame_types_if_defs.h`.

The *Frame body* consists of: the *Data header* which describes the datastream payload and the *Data body* which contains the actual datastream payload.

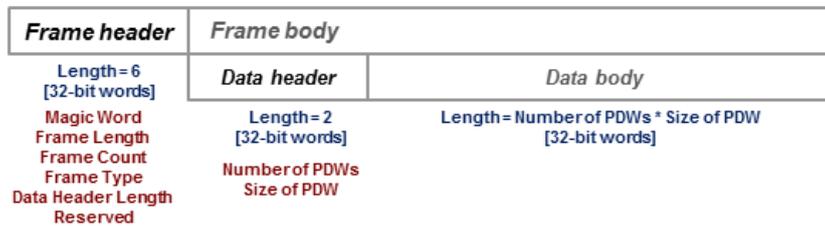


Fig. 11-1: PDW Data frame format

PDW Data Header

The PDW *Data header* structure, of type `struPDW_HEADER`, is described in the following table (total Data header length = 2 [32-bit words]).

Table 11-1: PDW Data header structure (`struPDW_HEADER`)

Word position in frame	Member name Member type	Description
7	nrOfPdws unsigned int	Number of PDWs contained in this frame
8	sizeOfPdw unsigned int	Size of PDW - the size in 32-bit words of a PDW

PDW Data Body

The PDW Data body is defined as an array of PDWs, of type `CPackedPDW`. The array size is given by **Number of PDWs** information in the *Data header*.

TOA Low in ns 32 Bit									
TOA High in 2 ³² ns 32 Bit									
Format 5 Bit			Center Frequency in kHz 27 Bit						
v	p	l	s	e	Rsvd ₂		Pulse Width in ns 25 Bit		
Frequency Shift / Bandwidth in kHz 20 Bit							Level in 0.1 dBµV or 0.1 dBµV/m 12 Bit		
io	DF Conf. in 0.1° 6 Bit			Modulation 5		Reserved 16 Bit			Sector 4 Bit
Pol ₂		DF Qual. in % 7		Elevation in 0.1° 11 Bit			Azimuth in 0.1° 12 Bit		
Channel 4 Bit			Reserved 28 Bit						

v: valid flag, p: pulse flag, l: LU flag, s: SNS flag, e: SNE flag, io: in/out flag

Fig. 11-2: PDW structure (word 0 to 7, bits 31 to 0)

Table 11-2: PDW data format description

Member name Member type	Description
m_u64TOA unsigned __int64	Time Of Arrival [ns] - Time is represented as a count of nanoseconds elapsed since 01.01.1970, 0:00 h UTC (<i>UTC as described in: "RECOMMENDATION ITU-R TF.460-5, STANDARD-FREQUENCY and TIME-SIGNAL EMISSIONS, 1970-1974-1978-1982-1986-1997"</i>).
m_u32Res_CFinKHz unsigned int	<ul style="list-style-type: none"> Bits #31 to #27 - PDW Format identifier Bits #26 to #0 - Center Frequency of signal in [kHz], max 134 [GHz] Field contains '0' if no valid Center Frequency could be determined.
m_u32Flags_Marks_PW unsigned int	<ul style="list-style-type: none"> Bit #31 - Valid Flag (1 - PDW is valid, 0 - PDW invalid) Bit #30 - Pulse Flag (1 - signal is a pulse, 0 - signal is CW and the signal was split into multiple PDWs) Bit #29 - Level Unit (LU) Flag (1 - level unit is [dBµV], 0 - level is a field strength in [dBµV/m]) Bit #28 - Signal no Start (SNS) Flag (1 - signal started before TOA of PDW, 0 - pulse start detected successfully) Bit #27 - Signal no End (SNE) Flag (1 - signal stops after TOA + PW of the PDW, 0 - pulse end detected successfully) Bits #26 to #25 - Reserved - Marker Bits for Blanking Bits #24 to #0 - Pulse Width (PW) in [ns] defined on 25-bit field, field contains 0 if no valid Pulse Width could be determined
m_u32BW_Level unsigned int	<ul style="list-style-type: none"> Bits #31 to #12 - Frequency Shift or Bandwidth of signal in [kHz] on 20-bit field, max 1 [GHz], Field contains '1048575' (i.e. binary all '1's) if no value could be determined Bits #11 to #0 - estimated Pulse Level in [dBµV] or Pulse Field Strength in [dBµV/m], (unit indicated by level unit flag on word 3) on 12-bit field. Range: [-200.0...200] in 0.1 steps, use sign extension. Range [-200.0 dBµV .. +200.0 dBµV] or [-200.0 dBµV/m .. +200.0 dBµV/m. Field contains '-204.8' (i.e. binary '1000 0000 0000') if no valid Level could be determined.

Member name Member type	Description
m_u32IO_DFE_Mod_Chip_Low unsigned int	<ul style="list-style-type: none"> Bit #31 - IO-Flag (1 - signal is inside Region of interest, 0 - signal is outside Region of Interest, unknown or inapplicable) Bits #30 to #25 - DF Azimuth Confidence. Range [0° - 6.2°] in 0.1° steps. Field contains '6.3' (i.e. binary '11 1111') if no valid DF Confidence could be determined. Bits #24 to #20 - Modulation ID as given by the enumeration ePDWModType. 00000: unknown 00001: unmodulated 00010: FM 00011: LFM 00100: PSK2 00101: PSK3 00110: PSK4 00111: PSKm 01000: NLFM 01001: SFM 01010: TFM 01011: pulse too short Others : reserved. Bits #19 to #4 - Reserved Bits #3 to #0 - Sector Reference antenna sector number of the measurement. If Sector is unknown or inapplicable then this field is set to 0.
m_u32POL_DFQ_EOA_AOA_Low unsigned int	<ul style="list-style-type: none"> Bits #31 to #30 - Polarity (00 - horizontal, 01 - vertical, 10 - left hand circular (counter-clockwise), 11 - right hand circular (clockwise)). If a device or an antenna in use does not support Polarization determination then this field is set to binary 00 Bits #29 to #23 - Direction Finding (DF) Quality. Range: [0-100%]. If no valid DF Quality could be determined then this field is set to 0. Bits #22 to #12 - estimated Elevation of signal. Range: [-90°, 90°] in 0.1° steps, use sign extension. Field contains '-102.4' (binary 100 0000 0000) if no valid Elevation could be determined. Bits #11 to #0 - estimated Azimuth of signal. Range: [0.0°, 359.9°] in 0.1° steps. Field contains '409.5' (binary 1111 1111 1111) if no valid Azimuth could be determined.
m_u32Reserved2_Low unsigned int	<ul style="list-style-type: none"> Bits #31 to #28 - Number of the Channel on which the pulse was detected. If no or only one channel is supported then this field is set to 0. Bits #27 to #0 - Reserved integer for future extensions. All bits set to 0

11.2 IQ Descriptor Words (IQDW) Datastream

This document concerns the data format description of the IQDW data packets provided by R&S receivers such as WPU500.

Two formats are supported:

- IQDW with 16-bit samples
- IQDW with 32-bit samples

In each case the AMMOS Frame Header and AMMOS Data Header are identical. Only the TX Datablock formats are different.

11.2.1 AMMOS Headers

Byte Number	Element	Description
AMMOS Frame Header (24 bytes)		
0x00 - 0x03	4 bytes Magic_Word	A word used to identify a Tx_Block (0xFB746572)
0x04 - 0x07	4 bytes Frame_Length	Overall Tx_Block length counted in 32-bit words
0x08 - 0x0B	4 bytes Frame_Count	A counter which counts the AMMOS frames in increasing order. This counter can be used to detect missing blocks. The counter starts with 0, when transmission is started.
0x0C - 0x0F	4 bytes Frame_Type	Used to identify the AMMOS frame type. 0x201 is used for IQDW data with 16-bit IQ samples.
0x10 - 0x13	4 bytes Data_Header_Length	Length of AMMOS data header counted in 32-bit words: 0x0C (12)
0x14 - 0x17	4 bytes Reserved	b[31:01] = reserved (0) b[0] = Mask for extraction 'frame size may exceed kFRH_FRAME_LENGTH_MAX'
AMMOS Data Header (48 bytes)		
0x18 - 0x1B	4 bytes ACH_Freq_Low	Current analysis channel frequency in Hz
0x1C - 0x1F	4 bytes ACH_Freq_High	
0x20 - 0x23	4 bytes ACH_Bandwidth	Current analysis channel 3dB bandwidth in Hz (with 0-padding)
0x24 - 0x27	4 bytes Samplerate	Sampling rate given in Hz Remark: The final sampling rate for the IQ-data is derived by: Samplerate * Interpolation / Decimation
0x28 - 0x2B	4 bytes Interpolation_Decimation	Interpolation/Decimation Factor referred to the ADC signal sample rate. Interpolation numerator (lower 16 bit) Decimation denominator (upper 16 bit)
0x2C - 0x2F	4 bytes Antenna_Voltage_Ref	Antenna reference voltage given in 0.1dBuV
0x30 - 0x33	4 bytes StartTime_Low	Indicates the reference time (StartTime) of the datastream. It is counted in nanoseconds relatively to the date 1.1.1970.
0x34 - 0x37	4 bytes StartTime_High	

Byte Number	Element	Description
0x38 - 0x3B	4 bytes SampleCount_Low	This counter indicates the sample number of the first sample from the data block relatively to the reference time (StartTime) of the data-stream. Time Stamp can be calculated with this formula: $\text{Timestamp [ns]} = \text{StartTime} + (\text{SampleCount} * 10e9 * \text{Decimation} / \text{Samplerate} * \text{Interpolation})$
0x3C - 0x3F	4 bytes SampleCount_High	
0x40 - 0x43	4 bytes K_Factor	kFactor - Correction factor of the current antenna, given in 0.1dB/m. Used to determine the field strength (in $\text{dB}\mu\text{V/m}$) at the antenna from the voltage level at the antenna input of the receiver. Contains antenna gain, cable attenuation, antenna switch matrix attenuation and anything else from air to antenna input. (the value 0x80000000 is used if no kFactor is defined).
0x44 - 0x47	4 bytes Datablock_Status_Word	b[31:16] = 0xFFFF b[15:02] = reserved b01 = blanking b00 = signal invalid
TX Datablock:		
See the following chapter for descriptions of the TX Datablocks in 16-bit and 32-bit formats		

11.2.2 TX Datablocks

Datablock with 16-bit samples

Byte Number	Element	Description
TX Datablock:		
0x48 – 0x67	32 bytes PDW	PDW of this IQDW. See PDW specification.
0x68 – 0x6B	4 bytes Samples	IQ samples used to calculate the above PDW. Upper 16-bit Imaginary component Lower 16-bit Real component
... IQ samples continued ... until Frame_Length * 4 bytes		

Datablock with 32-bit samples

Byte Number	Element	Description
TX Datablock:		
0x48 – 0x67	32 bytes PDW	PDW of this IQDW. See PDW specification.

Byte Number	Element	Description
0x68 – 0x6B	4 bytes Samples	IQ samples used to calculate the above PDW.
0x6C – 0x6F	4 bytes Sample (Imaginary component)	
... IQ samples continued ... until Frame_Length * 4 bytes		

A Extras

A.1 Data types definitions

The datastreams interface library defines the following global data types in the `rs_gx40x_p_types.h` file:

Type name	Description
<code>ptypBOOL</code>	32-bit type used for boolean data type. Values: <code>pkFALSE=0</code> and <code>pkTRUE=1</code>
<code>ptypUINT</code>	32-bit type used for unsigned integers (range: 0 to <code>pkUINT_MAX=4294967295</code>). The definition of this type ensures that it is represented on 32 bits independent of the used platform.
<code>ptypINT</code>	32-bit type used for signed integers (range: <code>pkINT_MIN=-2147483648</code> to <code>pkINT_MAX=2147483647</code>)
<code>ptypFLOAT_SP</code>	32-bit type used for IEEE 754/854 compatible floating point single precision formats.
<code>ptypPACKEDCOMPLEXI16</code>	32-bit type containing a complex number made up of two 16-bit (signed)integer numbers. The 16 most significant bits of the packed complex type contain the real part of the complex number. The 16 least significant bits of the packed complex type contain the imaginary part of the complex number
<code>ptypCHAR</code>	32-bit type containing one character . The 8 least significant bits of this 32-bit type contain one 8-bit character (range: <code>pkCHAR_MIN=-2147483648</code> to <code>pkCHAR_MAX=2147483647</code>)
<code>ptypPACKEDCHAR</code>	32-bit type containing up to four 8-bit characters packed together into a 32-bit word. The 8 least significant bits of this 32-bit word contain the first 8-bit character and so on. The type accepts only C-strings (null terminated).
<code>ptypPACKEDUNICODE</code>	32-bit type containing up to two 16-bit unicode characters packed together into a 32-bit word. The 16 least significant bits of this 32-bit word contain the first 16-bit character of the two characters. The type accepts only C-strings (null terminated).
<code>ptypBIGTIME</code>	64-bit type used for the representation of time, implemented as a union whose members vary on differing platforms. Time is represented as a count of microseconds elapsed since 01.01.1970, 0:00 h UTC (UTC as described in: "RECOMMENDATION ITU-R TF.460-5, STANDARD-FREQUENCY and TIME-SIGNAL EMISSIONS, 1970-1974-1978-1982-1986-1997"). Members for 32bit platform: <code>structTimeInTwoWords.uintTime_LoOrderBits</code> and <code>structTimeInTwoWords.uintTime_HiOrderBits</code> Members for 64bit platform: <code>uint64TimeInOneWord</code>

Type name	Description
<code>ptypBIGTIME_NS</code>	Same as above but here time is represented in nanoseconds .
<code>pmtypMSG_ENUM</code>	<p>represents a macro that is used for definition of enumeration types: <code>pmtypMSG_ENUM(enumerator, enumtyp)</code>. This macro ensures a 32-bit type enumeration value on all platforms.</p> <p>The enumeration data type (<code>enumtyp</code>) is based on an predefined enumeration (<code>enumerator</code>) if the enumerator values are represented as 32-bit numbers on the current platform:</p> <pre>typedef enum enumerator enumtyp</pre> <p>Otherwise the <code>enumtyp</code> will be defined as <code>ptypINT</code>:</p> <pre>typedef ptypINT enumtyp</pre>

A.2 File Types

File ending	File type	Description
<code>.dat</code>	AMREC recording	Datastream recording on an AMREC device (datastream type is specified in the "Data type" information field from the metadata). Each recording has associated information files that have the same file name as the recording but with different extensions (<code>.metadata</code> , <code>.idx</code> , <code>.bkm</code> , <code>.his</code>)
<code>.metadata</code>	Metadata of the recording	Record information such as data type, recording time, etc.
<code>.idx</code>	Index data of the recording	The index contains key data about the record that can be used for navigation in the recording
<code>.bkm</code>	Bookmarks of the recording	Bookmarks are identifiers in the recording data expressed as offsets from beginning of recording
<code>.his</code>	History data of the recording	Each successful call will result in a history comment related to the time when it has been made
<code>.riq</code>	Raw I/Q data	I/Q datastream in EB200 format
<code>.iq.tar</code>	Packed I/Q data	I/Q data files packed using <code>.tar</code> archiving format that are generated by Rohde & Schwarz® Spectrum Analyzers
<code>.aid</code>	AMMOS Intermediate (IF) data	Intermediate Frequency (baseband I/Q) datastream
<code>.adem</code> , <code>.aud</code>	AMMOS Analog Demodulator, Audio data	Audio datastream generated by an Analog Demodulator
<code>.daud</code>	AMMOS Digital Audio data	Audio datastream generated by a Digital Demodulator
<code>.asd</code>	AMMOS Spectrum data	FFT Spectrum datastream generated by a Fixed Frequency Processing module

File ending	File type	Description
.sym	AMMOS Symbol data	Demodulated Symbols datastream generated by a Digital Demodulator
.inst	AMMOS Instantaneous Data	Demodulated Instantaneous Data datastream generated by a Digital Demodulator
.img	AMMOS Image data	Image datastream generated by a Digital Demodulator or Decoder
.dtx	AMMOS Decoded Text data	Decoded Text datastream generated by a Decoder
.tsr	AMMOS Transmission System Result data	TSR datastream generated by a Digital Demodulator
.hist	AMMOS Histogram data	Histogram datastream generated by Frequency Hopping module
.hdw	AMMOS Hop Density Waterfall data	HDW datastream generated by Frequency Hopping module
.pdw	AMMOS Pulse Description Words	Pulse Description Words (PDW) generated by a WPU or TPA client
.ppdw	Pure Pulse Description Words	Continuous PDW stream without frame header information, i.e. not frame based as the .pdw format
.iqdw	I/Q data for Description Words	Contains the I/Q data corresponding to the Pulse Description Words of a .pdw file (generated along a .pdw file if PDW detail is needed)
.idx	Index data	Indexes of PDWs (start position and data length) in a .iqdw file (always generated along a .iqdw file)
.bin	Binary data	Contains bitstreams; used for importing bitstreams into CA250 from third party products
.bi8	Byte stream data	Contains symbol streams where each symbol consists of 8 bits; used for importing symbol streams into CA250 from third party products
.stx	Results data	Universal data format for CA250 result data (bitstreams, symbol streams, diagrams, decoded text, result tables)
.wav	Wave file	Waveform Audio File Format
.wv	Rohde & Schwarz® wave file	Wave audio format used by Rohde & Schwarz® Signal Generators
.apj	CA1xx Job file	Job settings in XML format
.gxjob	GX4xx Job file	Legacy format for Job settings in XML format (for GX430 up to version 2.82)
.isprj	CA100IS Project	Project settings in XML format
.js	JavaScript	Script file used for Automatic Processing function

File ending	File type	Description
.py	Python Script	Script file for the control of CA250
.xml	XML Description file	Used for different reports, configurations, layouts or list types
.cfg	XML Configuration file	Used for configuration descriptions
.pgxml	ParamGUI XML Description file	Used for GUI element descriptions
.QRSxml	XML Settings file	Used to save user specific settings of an application that will be used every time the application is started For CA250 - used to save session trackings and reports for connection to RAMON Report edit.
.wcxml	Waterfall colors	XML settings file
.csv	Comma Separated Value	Used for different log or result list types
.txt	ASCII text file	Used for different log, decoded result text types, or symbol streams from third party products
.log	Log file	Text file for different log types
.png	Portable Network Graphics	Bitmap image file format

Glossary

Symbols

μs: Microseconds

μV: Microvolts

A

ADC: Analog Digital Converter

AM: Amplitude Modulation

B

byte: Digital information unit with size = 8 bits

D

dB: Decibel

dB/m: Decibel per meter (for attenuation coefficients)

F

Fix: Fixed point - indicates a signed fixed point fractional number.

FM: Frequency Modulation

I

IF: Intermediate frequency

IM: The Imaginary part of a complex number

ISB: Independent Side Band

L

LSB: Lower Sideband

N

n/a: Not applicable

ns: Nanoseconds

P

ptypBIGTIME: Time representation [μs] data type (details: [chapter A.1, "Data types definitions"](#), on page 88).

ptypBIGTIME_NS: Time representation [ns] data type (details: [chapter A.1, "Data types definitions"](#), on page 88).

ptypBOOL: Boolean data type (details: [chapter A.1, "Data types definitions"](#), on page 88).

ptypCHAR: Character data type (details: [chapter A.1, "Data types definitions"](#), on page 88).

ptypFLOAT_SP: Floating point with single precision data type (details: [chapter A.1, "Data types definitions"](#), on page 88).

ptypINT: Integer data type (details: [chapter A.1, "Data types definitions"](#), on page 88).

ptypPACKEDCHAR: Packed characters data type (details: [chapter A.1, "Data types definitions"](#), on page 88).

ptypPACKEDUNICODE: Packed unicode characters data type (details: [chapter A.1, "Data types definitions"](#), on page 88).

ptypUINT: Unsigned integer data type (details: [chapter A.1, "Data types definitions"](#), on page 88).

R

RE: The Real part of a complex number

U

UL: Unsigned long

USB: Upper Sideband

Index

A

Audio Data	20
Data header	20
Frame structure	20
Sample formats	22

B

Burst Emission List Data	74
Data body	76
Data header	75
Frame structure	74

D

Datastream type	
ekFRH_DATASTREAM_AUDIODATA	20
ekFRH_DATASTREAM_BURST_EMISSIONS_LIST	74
ekFRH_DATASTREAM_DDCE_IFDATA_16RE_16IM_FIX	16
ekFRH_DATASTREAM_DDCE_IFDATA_32RE_32IM_FIX	16
ekFRH_DATASTREAM_DECODER_TEXT_DATA	49
ekFRH_DATASTREAM_EMISSION_LIST_DATA	70
ekFRH_DATASTREAM_HF_SCF_DATA	30
ekFRH_DATASTREAM_HF_SFF_DATA	30
ekFRH_DATASTREAM_HF_SSR_DATA	32
ekFRH_DATASTREAM_HF_TUNING_INDICATOR_DATA	29
ekFRH_DATASTREAM_HISTOGRAM_DATA	77
ekFRH_DATASTREAM_HOP_DENSITY_WATERFALL_DATA	79
ekFRH_DATASTREAM_IFDATA_16RE_16IM_FIX	8
ekFRH_DATASTREAM_IFDATA_16RE_16RE_FIX	8
ekFRH_DATASTREAM_IFDATA_32RE_32IM_FIX	8
ekFRH_DATASTREAM_IFDATA_32RE_32IM_FIX_RESCALED	8
ekFRH_DATASTREAM_IFDATA_32RE_32IM_FLOAT_RESCALED	8
ekFRH_DATASTREAM_IMAGEDATA	46
ekFRH_DATASTREAM_INSTANTANEOUSDATA	43
ekFRH_DATASTREAM_LEVELDATA	27
ekFRH_DATASTREAM_PULSE_DESCRIPTION_WORD_DATA	82
ekFRH_DATASTREAM_SCAN_LEVEL	24
ekFRH_DATASTREAM_SCAN_LEVEL_TUNING	24
ekFRH_DATASTREAM_SCAN_TUNING	24
ekFRH_DATASTREAM_SEGMENTATION_SPECIFICATION_FLOAT	37
ekFRH_DATASTREAM_SPECDATA_16BIT	34
ekFRH_DATASTREAM_SPECDATA_FLOAT	34
ekFRH_DATASTREAM_SPECTRALDETECTOR_DATA	72
ekFRH_DATASTREAM_SYMBOLDATA	39
ekFRH_DATASTREAM_TIMEDOMAIN_DATA	42
ekFRH_DATASTREAM_TRANSMISSION_SYSTEM_RESULT_DATA	51
ekFRH_DATASTREAM_TUNER_PIFPAN_DATA	28
Statistics Data	77

Decoded Text Data	49
Data header	50
Frame structure	49
Sample formats	50

E

EM010 SCF and SFF Scan Data	30
Data body	32
Data header	31
Frame structure	30
EM010 SSR Status Data	32
Data structure	33
Frame structure	32
EM010 Tuning Indicator Data	28
Datablock structure	29
Frame structure	29
Emission List Data	70
Data body	71
Data header	70
Frame structure	70

F

Frame format	5
Data Header Length	6
Frame body	6
Frame Count	6
Frame header: typFRH_FRAMEHEADER	5
Frame Length	6
Frame Type	6

H

Histogram Data	77
Data body	79
Data header	77
Frame structure	77
Hop Density Waterfall Data	79
Data body	81
Data header	80
Frame structure	79

I

IF Data	8
Data header	9
Datablock header	13
Frame structure	8
Sample formats	13
IF DDCE Data	16
Data header	17
Datablock header	18
Frame structure	16
Sample formats	19
Image Data	46
Data header	47
Frame structure	46
Sample formats	48
Instantaneous Data	43
Data body	45
Data header	44
Frame structure	43

P

PDW Data	82
Data header	82
Frame structure	82
Sample formats	83
ptyp Data types	88
Pulse Description Data	82, 84

S

Scan Data	24
Data body types	25
Data header	25
Datablock header	26
Frame structure	24
Sample formats	26
Segmentation Spectrum Data	37
Data header	37
Frame structure	37
Sample formats	38
Signal Level Indicator Data	27
Data structure	27
Frame structure	27
Spectral Detector List Data	72
Data body	73
Data header	72
Frame structure	72
Spectrum Data	34
Data header	34
Frame structure	34
Sample formats	36
Symbol Data	39
Data header	39
Frame structure	39
Sample format	41

T

Time Domain Data	42
Data body	43
Data header	42
Frame structure	42
Transmission System Result Data	51
ACARS	63
ASCII	54
ATIS	54
CLOVER	66
Data body	53
Data header	51
F7W	67
FMS-BOS	54
Frame structure	51
MPT1327	62
PACKET_RADIO	59
PACTOR	64
PDU	69
POCSAG	58
UNICODE	54
ZVEI	56
ZVEI-VDEW	57
Tuner PIF Panorama Data	28
Frame structure	28