Programmer Manual

Tektronix

Tektronix Protocol Testers

Record File API Description

C73000-M6076-C495-1

Tektronix Berlin GmbH & Co. KG
Wernerwerkdamm 5
13629 Berlin
GERMANY

Contents

Preface

With the record file API it is possible to write records in K15 format and to read record files of the K1205, the K1103 and the K1297 Protocol Testers. You can use this tool for analysis, statistics, filtering, converting and processing of existing record files.

This DLL is usable under Microsoft Windows XP, Windows 2000, Windows NT, or Windows 98 or 95. This documentation is designed for the programming languages C and C++.

Installation

List of Files

The following files are part of the installation in the *RecFileAPI* directory:

- \K12RecFl.dll

  This DLL that makes the functions for reading and writing record files available. Place the DLL into the *windows\\system* directory (Windows 9x) or in the *%systemroot%\\system32* directory (Windows NT).

- \K12RecFl.lib

  Library for VC 6.0 that automatically tries to load the DLL. You can link this library to your project.

- *Lib_VC42\\K12RecFl.lib*

  Library for VC 4.2 that automatically tries to load the DLL. You can link this library to your project.

- \header\\K12RecFl.h

  Header file that defines values needed as parameters of some DLL functions and the structure of events. You have to include this header into your source-code.

- \sample\\ListEvt.cpp

  Demo that shows how to use the DLL (lists all events of an given record file).

How to Install

1. Copy the DLL into the Windows system directory
   (such as *system32* for Windows XP and *system32* for Windows NT
   and *Windows\\system* for Windows 98 and 95).

2. Copy the header files into your project directory or into the header directory of your compiler.

3. If you want to use it, add the *.lib* file to your compiler library directory.

Used Data Types

Table 1: Used data types

Data Type	Length
integer	32 bit signed
long	32 bit signed
unsigned long	32 bit unsigned
unsigned short	16 bit unsigned
unsigned char	8 bit unsigned

Default Settings

Table 2: Parameters and default values

Parameter	Possible values	Default value
Swap Mode	K12_REC_NO_SWAP (0) K12_REC_AUTO_SWAP (1) K12_REC_DO_SWAP (2)	K12_REC_AUTO_SWAP (1)
Timestamp Mode	K12_REC_DO_NOT_CALC (0) K12_REC_DO_CALC (1)	K12_REC_DO_NOT_CALC (0)

Limitations

With the record file tool it is possible to open (read open or write open) up to 100 files at the same time.

The size of one record file is limited to 2 GByte.

Event Structures

The events in a record file are always stored in a certain fixed sequence:

- The first events in the file are the configuration events specifying the configuration of the hardware. The first configuration event is an LDS (logical data source) followed by one or more LL (logical link) configuration events.

  Within these configuration events, cross-references to other files (for example stack files) may appear. The interpretation of this data depends upon the availability of the referenced files. Subsequent changes to the referenced files thus entails a new interpretation of the configuration events.

- After the configuration events, frame, signaling and information events may follow in any arbitrary sequence. The source of frame and signaling events is the data measured by the hardware measuring devices.

Protocol Tester applications store configurations, text events, frame events and signaling events.

Use the *Get..(..)* commands to access events (see chapter *Command Set*, section *Reading Commands*). These functions always return a pointer to an event header (see section *Event Header*). From the header you can extract the event type contained behind the structure.

The following sections describe the event types. The structures are defined in the header file *event.h*.

Event Header

The event header is placed at the beginning of all events (returned by the *Get...()* functions). With this header it is possible to check, how long the event is and what the *event* type is. The structure of the event header is shown below.

```
typedef struct
{
   unsigned long   ulLength;
   unsigned short  usGroup;
   unsigned short  usType;
} K12_REC_stEventHead_t;
```

ulLength (offset: 0x0)

This variable contains the total size of the event, including this header with this size variable and the number of padding bytes.

usGroup (offset: 0x04) / usType (offset: 0x06)

The values of *usGroup* and *usType* are shown in Table 3.

Table 3: Event types

usGroup	usType	Name of event
K12_REC_EVENTGRP_DATA (0x0001)	K12_REC_EVENTTYP_FRAME (0x0020)	Frame event (normal HDLC data)
K12_REC_EVENTGRP_DATA (0x0001)	K12_REC_EVENTTYP_TRANS (0x0021)	Transparent frame
K12_REC_EVENTGRP_DATA (0x0001)	K12_REC_EVENTTYP_BIT (0x0022)	Bit data (TRAU frame)
K12_REC_EVENTGRP_TEXTL1 (0x0002)	K12_REC_EVENTTYP_TEXT (0x0030)	Text event
K12_REC_EVENTGRP_TEXTL1 (0x0002)	K12_REC_EVENTTYP_L1 (0x0031)	L1 event
K12_REC_EVENTGRP_TEXTL1 (0x0002)	K12_REC_EVENTTYP_L1BAI (0x0032)	L1 event (BAI)
K12_REC_EVENTGRP_TEXTL1 (0x0002)	K12_REC_EVENTTYP_L1VX (0x0033)	L1 event (VX)
K12_REC_EVENTGRP_RFCONF (0x0007)	K12_REC_EVENTTYP_LDS_CONF (0x0040)	LDS (Logical Data Source) configuration event
K12_REC_EVENTGRP_RFCONF (0x0007)	K12_REC_EVENTTYP_LL_CONF (0x0041)	LL (Logical Link) configuration event
K12_REC_EVENTGRP_DATA (0x0001)	K12_REC_EVENTTYP_FRAG (0x0024)	With Frame–, Bit–, and Transparent–Events: used to mark the frame which is a fragment (bit 3)
K12_REC_EVENTGRP_DATA (0x0001)	K12_REC_EVENTTYP_BITFRAG (0x0026)	With Frame–, Bit–, and Transparent–Events: used to mark the frame which is a fragment (bit 3)
K12_REC_EVENTGRP_DATA (0x0001)	K12_REC_EVENTTYP_GEN (0x0028)	With Frame–, Bit–, and Transparent–Events: used to mark the frame which is generated by the LSA (bit 4)

Table 3: Event types (Cont.)

usGroup	usType	Name of event
K12_REC_EVENTGRP_DATA (0x0001)	K12_REC_EVENTTYP_BITGEN (0x002A)	With Frame–, Bit–, and Transparent–Events: used to mark the frame which is generated by the LSA (bit 4)
K12_REC_EVENTGRP_RFCONF (0x0007)	K12_REC_EVENTTYP_LDS_CONF (0x0040)	Recordingfile–Config–Events: LDS (Logical Data Source) configuration
K12_REC_EVENTGRP_RFCONF (0x0007)	K12_REC_EVENTTYP_LL_CONF (0x0041)	Recordingfile–Config–Events: LL (Logical Link) configuration

Event Types

Configuration Events

Configuration events are used to store the settings of the measurement device. There are two types of configuration events: LDS (logical data source) configuration events and LL (logical link) configuration events.

LDS Configuration Events

Group 0x007 / Type 0x0040

(symbolic:
K12_REC_EVENTGRP_RFCONF / K12_REC_EVENTTYP_LDS_CONF)

LDS (Logical Data Source) configuration events are used to group the configuration events: A configuration always starts with an LDS followed by one or more LL configuration events. The LDS keeps the number of LL configuration events. It stores a unique (in the record file) ID. The ID is also part of each LL configuration event, that belongs to this LDS.

```
typedef struct
{
   K12_REC_stEventHead_t         eh;
   unsigned long                 ulLDSId;
   unsigned long                 ulNumberOfLLs;
   char                          czLdsName[4];
} K12_REC_stEventLDSConfig2_t;
```

ulLDSId (offset: 0x08)

An unique label of the LDS (logical data source) used for the measurement.

ulNumberOfLLs (offset: 0x0C)

The number of LL configuration events, which belong to this LDS.

czLdsName[4] (offset: 0x0D)

The name of the LDS (\0 terminated).

LL Configuration Events

Group 0x007 / Type 0x0041

(symbolic:
K12_REC_EVENTGRP_RFCONF / K12_REC_EVENTTYP_LL_CONF)

An LL (logical link) configuration event stores the settings of a logical link that was created by the measurement device. It contains the source of the signal (Board, Port), the names of the links (plain text), and the decoding stack used.

```
typedef struct
{
   K12_REC_stEventHead_t        eh;
   unsigned long                ulLDSId;
   unsigned long                ulSymLLId;
   unsigned long                ulColor;
   unsigned short               usSystemId;
   unsigned short               usBoardType;
   unsigned char                ucBoardId;
   unsigned char                ucPortNr;
   unsigned char                ucPortType;
   unsigned char                ucLLType;
   unsigned short               usVarPartLen;
   unsigned short               usHWPartLen;
   unsigned short               usLLNameLen;
   unsigned short               usStackPathLen;
   char                         cVarData[4];
} K12_REC_stEventLLConfig3_t;
```

ulLDSId (offset: 0x08)

ID of the corresponding LDS (logical data source).

ulSymLLId (offset: 0x0c)

Symbolic LL (Logical Link) ID: Each Logical Link that has been created has his unique ID specifying the relationship between an LL configuration event and the Data–Events. This ID is registered in ‘ulOrigin’ of each data event that belongs to this LL configuration event (Frame–, Bit– and Transparent–Events).

ulColor (offset: 0x10)

Foreground and background color of the logical link when it is displayed in the monitor window. Bits 0–7 represent the background color, bits 8–15 the foreground color. The values of the supported colors are shown in Table 4.

Table 4: Color values

Color	Value
black	K12_REC_COL_BLACK (0x00)
blue	K12_REC_COL_BLUE (0x01)
green	K12_REC_COL_GREEN (0x02)
cyan	K12_REC_COL_CYAN (0x03)
magenta	K12_REC_COL_MAGENTA (0x05)
brown	K12_REC_COL_BROWN (0x06)
light gray	K12_REC_COL_LT_GRAY (0x07)
gray	K12_REC_COL_GRAY (0x08)
light blue	K12_REC_COL_LT_BLUE (0x09)
light green	K12_REC_COL_LT_GREEN (0x0A)
light cyan	K12_REC_COL_LT_CYAN (0x0B)
light magenta	K12_REC_COL_LT_MAGENTA (0x0D)
yellow	K12_REC_COL_YELLOW (0x0E)
white	K12_REC_COL_LT_WHITE (0x0F)

usSystemId (offset: 0x14)

ID of the device responsible for the measurement (Tektronix internal use).

usBoardType (offset: 0x16)

Indicates what kind of measurement-board is related with the Logical Link

Table 5: Board values

Board type	Value
unknown	K12_REC_BOARD_UNKNOWN (0x0000)
PRIMO	K12_REC_BOARD_PRIMO (0x0001)
BAI	K12_REC_BOARD_BAI (0x0002)
PRIMO 32	K12_REC_BOARD_PRIMO_32 (0x0003)
AP	K12_REC_BOARD_AP (0x0004)
PRIME	K12_REC_BOARD_PRIME (0x0005)
ATM	K12_REC_BOARD_ATM (0x0006)
AP4	K12_REC_BOARD_AP4 (0x0007)
PC	K12_REC_BOARD_PC (0x0008)
ETHERNET	K12_REC_BOARD_ETHERNET (0x0009)
ATM PowerAAL	K12_REC_BOARD_MIDAS (0x000A)
ATM PCE	K12_REC_BOARD_PCE (0x000B)
PowerWAN	K12_REC_BOARD_POWERWAN (0x000C)

ucBoardId (offset: 0x18)

The BoardID is the number of the measurement–board in the device. This number is necessary because there can be more than one measurement board in the device. It is used to identify from which board the data came.

ucPortNr (offset: 0x19)

The number of the port on the measurement board the data came from.
Port A = 0,
Port B = 1,
...

ucPortType (offset: 0x1a)

Describes the kind of port the data came from.

Table 6: Port type value

Port type	Value
unknown	K12_REC_PORT_UNKNOWN (0x00)
PRIMO	K12_REC_PORT_PRIMO (0x01)
S0	K12_REC_PORT_S0 (0x02)
U2B1Q	K12_REC_PORT_U2B1Q (0x03)
Offline	K12_REC_PORT_OFFLINE (0x04)
E1	K12_REC_PORT_E1 (0x05)
T1	K12_REC_PORT_T1 (0x06)
ATM E1/T1	K12_REC_PORT_ATME1T1 (0x07)
ATM E3/T3	K12_REC_PORT_ATME3T3 (0x08)
ATM STM1E	K12_REC_PORT_ATMSTM1E (0x09)
ATM STM1O	K12_REC_PORT_ATMSTM1O (0x0A)
ATM STM4	K12_REC_PORT_ATMSTM4 (0x0B)
ATM IBM25	K12_REC_PORT_ATMIBM25 (0x0C)
VX	K12_REC_PORT_VX (0x0D)
PRIMO2	K12_REC_PORT_PRIMO2 (0x0E)
DS0	K12_REC_PORT_DS0 (0x0F)
PRIMO_NEW	K12_REC_PORT_PRIMO_NEW (0x10)
ETHERNET	K12_REC_PORT_ETHERNET (0x11)
AB	K12_REC_PORT_AB (0x12)
AB_NET	K12_REC_PORT_AB_NET (0x13)
PCE: ATM2_E1DS1	K12_REC_PORT_ATM2_E1DS1 (0x14)
PCE: ATM2_E3DS3	K12_REC_PORT_ATM2_E3DS3 (0x15)
PCE: ATM2_STM1EL	K12_REC_PORT_ATM2_STM1EL (0x16)
PCE: ATM2_STM1OP	K12_REC_PORT_ATM2_STM1OP (0x17)
POWERWAN	K12_REC_PORT_POWERWAN (0x18)

ucLLType (offset: 0x1b)

This shows the mode, in which the Logical Link works.

usVarPartLen (offset: 0x1c)

The length of the variable part of an LL configuration event. The beginning of the variable part is cVarData[0], the end is cVarData[usVarPartLen – 5].

usHWPartLen (offset: 0x1e)

The hardware parameters are a content of the variable part of an LL configuration event. The beginning of the hardware part is cVarData[0], the end is cVarData[usHWPartLen – 1]. The hardware part must end with *0*.

usLLNameLen (offset: 0x20)

The LL name is the string that is displayed in the monitor window. It is just as the hardware parameter part of the variable part of the LL configuration event. The beginning of the LL name is cVarData[usHWPartLen], the end is cVarData[usHWPartLen + usLLNameLen – 1]. LL name must end with *0*.

usStackPathLen (offset: 0x22)

The stack path is a string that gives the name of the stack with which the recording should be decoded. It is also a content of the variable part of the LL configuration event. The beginning of the stack path is
cVarData[usHWPartLen + usLLNameLen].
The end is cVarData[usHWPartLen + usLLNameLen + usStackPathLen – 1]. The stack path must end with *0*.

cVarData[4] (offset: 0x24)

Stores four bytes of the variable part of the LL configuration event.

Data Events

Frame Events

Group 0x001 / Type 0x0020 (frame data)
Group 0x001 / Type 0x0024 (fragmented frame data)
Group 0x001 / Type 0x0028 (frame data generated by the LSA)

(symbolic:
K12_REC_EVENTGRP_DATA / K12_REC_EVENTTYP_FRAME)

This event stores data frames that were measured by the hardware, for example by a E1/DS-1 board.

```
typedef struct
{
   K12_REC_stEventHead_t          eh;
   unsigned long                  ulFrameLength;
   unsigned long                  ulOrigin;
   unsigned long                  ulStatus;
   unsigned long                  ulFrameId;
   unsigned long                  ulTimestampHigh;
   unsigned long                  ulTimestampLow;
   unsigned char                  ucData[4];
} K12_REC_stEventFrame_t;
```

ulFrameLength (offset: 0x08)

Frame (data) length in bytes.

The frame dData is stored in the data[4] field in the structure. The data begins at data[0] and ends at data[ulFrameLength - 1].

ulOrigin (offset: 0x0c)

The ID of the logical link the data came from. See section *LL configuration events*.

ulStatus (offset: 0x10)

Frame status (not defined yet).

ulFrameId (offset: 0x14)

The unique number of this frame from one origin.

ulTimestampHigh (offset: 0x18)

Bits 32–64 of the timestamp.

ulTimestampLow (offset: 0x1c)

Bits 0–31 of the timestamp.

ucData[4] (offset: 0x20)

Stores four bytes of the data event.

Transparent Frame Events

Group 0x001 / Type 0x0021

(symbolic: K12_REC_EVENTGRP_DATA / K12_REC_EVENTTYP_TRANS)

The container for transparent data is the same as for frame data.

Bit-Frame Events

Group 0x001 / Type 0x0022 (bit data)
Group 0x001 / Type 0x0026 (fragmented bit data)
Group 0x001 / Type 0x002A (bit data generated by the LSA)

(symbolic: K12_REC_EVENTGRP_DATA / K12_REC_EVENTTYP_BIT)

```
typedef struct
{
   K12_REC_stEventHead_t        eh;
   unsigned long                ulFrameLength;
   unsigned long                ulOrigin;
   unsigned long                ulStatus;
   unsigned long                ulBitLength;
   unsigned long                ulTimestampHigh;
   unsigned long                ulTimestampLow;
   unsigned char                ucData[4];
} K12_REC_stEventBitFrame_t;
```

ulFrameLength (offset: 0x08)

Frame(data) length in bytes.

The frame data is stored in the data[4] field in the structure. The data field begins at data[0] and ends at data[ulFrameLength - 1].

ulOrigin (offset: 0x0c)

The ID of the logical link the data came from. See section *LL configuration events*.

ulStatus (offset: 0x10)

Frame status (not defined yet).

ulBitLength (offset: 0x14)

Frame (data) length in **bits**.

The frame data is stored in the data[4] field in the structure. The data field begins at data[0] and ends at data[ulBitLength / 8].

ulTimestampHigh (offset: 0x18)

Bits 32–64 of the timestamp.

ulTimestampLow (offset: 0x1c)

Bits 0–31 of the timestamp.

ucData[4] (offset: 0x20)

Stores the data event.

Text and Layer 1 Events

Text Events

Group 0x002 / Type 0x0030

(symbolic:
K12_REC_EVENTGRP_TEXTL1 / K12_REC_EVENTTYP_TEXT)

This event stores messages that were created by Protocol Tester applications. Text events are used for information within the record file. For example: When a recording is started, a text–event is created to indicate the start time.

```
typedef struct
{
   K12_REC_stEventHead_t        eh;
   unsigned long                ulTextLength;
   unsigned long                ulOrigin;
   unsigned long                ulTextType;
   unsigned long                ulFrameId;
   unsigned long                ulTimestampHigh;
   unsigned long                ulTimestampLow;
   char                         cText[4];
} K12_REC_stEventTextFrame_t;
```

ulTextLength (offset: 0x08)

Text–length in chars (without \0)

The text is stored in the cText[4] field in the structure. The text begins at data[0] and ends at data[ulTextLength].

ulOrigin (offset: 0x0c)

The ID of the logical link the data came from. See section *LL configuration events*.

ulTextType (offset: 0x10)

Frame status (not defined yet).

ulFrameId (offset: 0x14)

The unique number of this frame from one origin.

ulTimestampHigh (offset: 0x1c)

Bits 32–64 of the timestamp.

ulTimestampLow (offset: 0x20)

Bits 0–31 of the timestamp.

cText[4] (offset: 0x24)

Stores the text event.

Layer 1 Events

Group 0x002 / Type 0x0031

(symbolic: K12_REC_EVENTGRP_TEXTL1 / K12_REC_EVENTTYP_L1)

Group 0x002 / Type 0x0032

(symbolic:
K12_REC_EVENTGRP_TEXTL1 / K12_REC_EVENTTYP_L1BAI)

Group 0x002 / Type 0x0033

(symbolic: K12_REC_EVENTGRP_TEXTL1 / K12_REC_EVENTTYP_L1VX)

Layer 1 events are used to give a status of the physical layer of the protocol. They are generated by the measuring hardware whenever error conditions are met.

```
typedef struct
{
   K12_REC_stEventHead_t        eh;
   unsigned long                ulL1EventType;
   unsigned long                ulEventValue;
   unsigned long                ulErrorCount;
   unsigned long                ulFrameId;
   unsigned long                ulTimestampHigh;
   unsigned long                ulTimestampLow;
   unsigned short               usSystemId;
   unsigned short               usBoardType;
   unsigned char                ucBoardId;
   unsigned char                ucPortNr;
   unsigned char                ucPortType;
   unsigned char                ucDummy;
} K12_REC_stEventL1Frame_t;
```

ulL1EventType (offset: 0x08)

Gives cause of the Layer 1 failure. See Table 7.

Table 7: Layer 1 Event failure values

L1 Event	Value
CRC error	K12_REC_L1ERR_CRC (1)
Frame error	K12_REC_L1ERR_FRAME (2)
Multiframe error	K12_REC_L1ERR_MULTIFRAME (3)
Special CEPT error	K12_REC_L1ERR_CEPT (4)

Table 7: Layer 1 Event failure values (Cont.)

L1 Event	Value
Bipolar violation	K12_REC_L1ERR_VIOLATION (5)
Loss of physical signal level	K12_REC_L1ERR_SIGNAL_LOSS (21)
PLL not locked	K12_REC_L1ERR_PLL (22)
Red alarm condition	K12_REC_L1ERR_RED_ALARM (23)
Yellow alarm condition	K12_REC_L1ERR_YELLOW_ALARM (24)
Back to normal	K12_REC_L1ERR_BACK_TO_NORM (25)
Blue alarm condition	K12_REC_L1ERR_BLUE_ALARM (26)
PRI error overflow shut down	K12_REC_L1ERR_PRI_SHUT_DOWN (27)
PRI buffer over/underflow	K12_REC_L1ERR_PRI_BUFFER (28)
Loss of multiframe alignment	K12_REC_L1ERR_ALIGN_LOSS (29)
Bipolar violation generated	K12_REC_L1ERR_VIOLATION_GEN (110)
CRC error generated	K12_REC_L1ERR_CRC_GEN (111)
Frame error generated	K12_REC_L1ERR_FRAME_GEN (112)
Multiframe error generated	K12_REC_L1ERR_MULTIFRAME_GEN (113)
Special CEPT error generated	K12_REC_L1ERR_CEPT_GEN (114)

ulEventValue (offset: 0x0c)

Tektronix internal use.

ulErrorCount (offset: 0x10)

Tektronix internal use.

ulFrameId (offset: 0x14)

The unique number of this frame.

ulTimestampHigh (offset: 0x18)

Bits 32–64 of the timestamp.

ulTimestampLow (offset: 0x1c)

Bits 0–31 of the timestamp.

usSystemId (offset: 0x20)

ID of the device responsible for the measurement: Tektronix internal use.

usBoardType (offset: 0x22)

Indicates what kind of measurement board is related to the logical link.
See table 8.

Table 8: Board type value

Board type	Value
unknown	K12_REC_BOARD_UNKNOWN (0x0000)
PRIMO	K12_REC_BOARD_PRIMO (0x0001)
BAI	K12_REC_BOARD_BAI (0x0002)
PRIMO 32	K12_REC_BOARD_PRIMO_32 (0x0003)
AP	K12_REC_BOARD_AP (0x0004)
PRIME	K12_REC_BOARD_PRIME (0x0005)
ATM	K12_REC_BOARD_ATM (0x0006)
AP4	K12_REC_BOARD_AP4 (0x0007)
PC	K12_REC_BOARD_PC (0x0008)
ETHERNET	K12_REC_BOARD_ETHERNET (0x0009)
ATM PowerAAL	K12_REC_BOARD_MIDAS (0x000A)
ATM PCE	K12_REC_BOARD_PCE (0x000B)
PowerWAN	K12_REC_BOARD_POWERWAN (0x000C)

ucBoardId (offset: 0x24)

The Board ID is the number of the measurement board in the device. This number is necessary because there can be more than one measurement board in the device. It is used to identify the board from which the data came.

ucPortNr (offset: 0x25)

The number of the port on the measurement board, the data came from.
Port A = 0,
Port B = 1
...

ucPortType (offset: 0x26)

The kind of port the data came from. See Table 9.

Table 9: Port type value

Port type	Value
unknown	K12_REC_PORT_UNKNOWN (0x00)
PRIMO	K12_REC_PORT_PRIMO (0x01)
S0	K12_REC_PORT_S0 (0x02)
U2B1Q	K12_REC_PORT_U2B1Q (0x03)
Offline	K12_REC_PORT_OFFLINE (0x04)
E1	K12_REC_PORT_E1 (0x05)
T1	K12_REC_PORT_T1 (0x06)
ATM E1/T1	K12_REC_PORT_ATME1T1 (0x07)
ATM E3/T3	K12_REC_PORT_ATME3T3 (0x08)
ATM STM1E	K12_REC_PORT_ATMSTM1E (0x09)
ATM STM1O	K12_REC_PORT_ATMSTM1O (0x0A)
ATM STM4	K12_REC_PORT_ATMSTM4 (0x0B)
ATM IBM25	K12_REC_PORT_ATMIBM25 (0x0C)
VX	K12_REC_PORT_VX (0x0D)
PRIMO2	K12_REC_PORT_PRIMO2 (0x0E)
DS0	K12_REC_PORT_DS0 (0x0F)
PRIMO_NEW	K12_REC_PORT_PRIMO_NEW (0x10)
ETHERNET	K12_REC_PORT_ETHERNET (0x11)
AB	K12_REC_PORT_AB (0x12)
AB_NET	K12_REC_PORT_AB_NET (0x13)
PCE: ATM2_E1DS1	K12_REC_PORT_ATM2_E1DS1 (0x14)
PCE: ATM2_E3DS3	K12_REC_PORT_ATM2_E3DS3 (0x15)
PCE: ATM2_STM1EL	K12_REC_PORT_ATM2_STM1EL (0x16)
PCE: ATM2_STM1OP	K12_REC_PORT_ATM2_STM1OP (0x17)
POWERWAN	K12_REC_PORT_POWERWAN (0x18)

ucDummy (offset: 0x27)

Dummy field for alignment.

Event Trailer

Note. Existing trailer structures are modified and new trailers are added on short term notice. Whilst the information is prepared in good faith, no warranty is given as to its accuracy or completeness. Tektronix reserves the right to change the contents of this document in any way for any reason and at any time.

The Event Trailer may exist in events of type K12_REC_stEventFrame_t and contains specific data, for example from ATM connections or Frame Processing Methods (FPM). This data is appended directly to the event frame data and is part of the Event data.

The existence of an Event Trailer and its length can be calculated using the two length values ulLength in structure K12_REC_stEventHead_t and ulFrameLength in structure K12_REC_stEventFrame_t:

FPM Trailer length = eh.ulLength – ulFrameLength – 32

with
32 = sizeof(K12_REC_stEventFrame_t) – sizeof(K12_REC_stEventFrame_t.ucData)

C syntax notation:
K12_REC_stEventFrame_t *pFrame;
TrailerLen = pFrame–>eh.ulLength – (sizeof(K12_REC_stEventFrame_t) – 4) – pFrame–>ulFrameLength;

Each Event Trailer consists of a common and specific trailer section.

The offset used in the following sections refer to the beginning of the Event Trailer and not to the Event.

Note. Trailers have a minimum length of 8 bytes. A calculated *TrailerLen* of less then 8 can be ignored.

Some trailers are aligned and some are not aligned. Not aligned trailers begin direct after frame data. Aligned trailers begin on a *long*-aligned address (address can be divided by 4). There may be 1 to 3 bytes unused data between frame data and trailer.

Common Section

unsigned short usTrailerType (offset 0x0)

This value defines the FPM Trailer type:

0x10 UMTS FP Iub/Iur data frame (not handled by reassembler)

0x11 UMTS FP Iub/Iur control frame

0x30 UMTS FP Iub/Iur data frame (handled by reassembler)

0x50 UMTS FP Iu UP data frame

0x80 ATM connection data

0x91 UMTS FP Iub/Iur reasembled data frame (creator: reassembler/LSA, first version)

0x92 UMTS FP Iub/Iur reasembled data frame (creator: reassembler/LSA)

unsigned short usVersion (offset 0x2)

Each type of trailer (see above) may have its own version control.
For example, for UMTS FP Iub/Iur the following usVersion values are used:

0	K1297-G20 V2.01
1	K1297-G20 V2.01.04 (Patch)
2	K1297-G20 V2.40
3	K1297-G20 V2.50

Note. The ATM connection data (usTrailerType 0x80) has only one version, that is 1.

unsigned long ulLength (offset 0x4)

The length of the Event Trailer (supported since usVersion 1)

ATM Connection Data

usTrailerType=0x80
usVersion=1

The usTrailerType 0x80 indicates the following ATM connection data.

Note. This trailer is created by the ATM device software running on PCE and PowerAAL boards. If any Frame Processing Method (FPM) is configured for an ATM connection, the appropriate FPM Trailer is created instead of this ATM connection data trailer.

unsigned short usVpi (offset 0x8)

Virtual Path Identifier

unsigned short usVci (offset 0xa)

Virtual Channel Identifier

unsigned char   ucCid (offset 0xc)

Channel Identifier

unsigned char   ucAalType (offset 0xd)

ATM Adaptation Layer Type:

0	none
1	AAL–2 path
2	AAL–2 SSSAR
3	AAL–2 CPS
4	AAL–2 SSTED
5	AAL–5
6	AAL–0

unsigned char   ucDirection (offset 0xe)

ATM connection direction:

1	Rx
2	Tx
3	RxTx

unsigned char   ucMultiPvcGroup (offset 0xf)

internally used M–PVC group number, 0=none

UMTS FP Iub/Iur Data Frame Trailer Section (V. 0 or 1)

usTrailerType=0x10 (not handled by reassembler)
or
usTrailerType=0x30 (handled by reassembler)
usVersion= 0 or 1

unsigned short usVPI (offset 0x8)

Virtual Path Identifier

unsigned short usVCI (offset 0xa)

Virtual Channel Identifier

unsigned char ucCID (offset 0xc)

Sub–Channel Identifier

unsigned char ucIfID (offset 0xd)

Interface Identifier

unsigned char ucRadioMode (offset 0xe)

Radio mode:

1	FDD
2	TDD 3.84 Mcps
3	TDD 1.28 Mcps

unsigned char ucDirection (offset 0xf)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned short usCFN (offset 0x10)

Connection Frame Number

unsigned char ucTrChType (offset 0x12)

Type of Transport Channel:

1	Broadcast Channel (BCH)
2	Paging Channel (PCH)
3	Common Packet Channel (CPCH)
4	Random Access Channel (RACH)
5	Forward Link Access Channel (FACH)
6	Uplink Shared Channel (USCH)
7	Downlink Shared Channel (DSCH)
8	Dedicated Channel (DCH)

unsigned char ucTFI (offset 0x13)

Transport Format: Index

unsigned long ulTrBlockSize (offset 0x14)

Transport Format: Transport Block Size (Bits)

unsigned long ulTrBlockSetSize (offset 0x18)

Transport Format: Transport Block Set Size (Bits)

unsigned long ulTTI (offset 0x1c)

Transport Format: Transmission Time Interval

unsigned char ucErrCodeType (offset 0x20)

Transport Format: Error Protection Code Type:

1	No Coding
2	Turbo Coding
3	Conventional Coding

unsigned char ucErrCodeRate (offset 0x21)

Transport Format: Error Protection Code Rate:

2	Half (1/2)
3	Third (1/3)

unsigned short usErrCodeMatching (offset 0x22)

Transport Format: Static Rate Matching Parameter

unsigned long ulCRCSize (offset 0x24)

Transport Format: Size of CRC

unsigned char ucLgChType

(10 entries: offsets 0x28, 0x2c, 0x30, 0x34, 0x38, 0x3c, 0x40, 0x44, 0x48, 0x4c)

Mapped Logical Channels (10 entries): Type of Logical Channel:

1	Broadcast Control Channel (BCCH)
2	Paging Control Channel (PCCH)
3	Dedicated Control Channel (DCCH)
4	Common Control Channel (CCCH)
5	Common Traffic Channel (CTCH)
6	Dedicated Traffic Channel (DTCH)
7	Shared Channel Control Channel (SHCCH)

unsigned char ucRlcMode

(10 entries: offsets 0x29, 0x2d, 0x31, 0x35, 0x39, 0x3d, 0x41, 0x45, 0x49, 0x4d)

Mapped Logical Channels (10 entries): RLC Mode:

1	Transparent Mode, segmented
2	Transparent Mode, non–segmented
3	Unacknowledged Mode
4	Acknowledge Mode

unsigned char ucLI

(10 entries: offsets 0x2a, 0x2e, 0x32, 0x36, 0x3a, 0x3e, 0x42, 0x46, 0x4a, 0x4e)

Mapped Logical Channels (10 entries): Length of LI field (7 or 15)

unsigned char ucCTid

(10 entries: offset 0x2b, 0x2f, 0x33, 0x37, 0x3b, 0x3f, 0x43, 0x47, 0x4b, 0x4f)

Mapped Logical Channels (10 entries): Identification of Logical Channel

(0...14; 255 for definite logical channel without CT id)

unsigned char ucNumLgChMapd (offset 0x50)

Number of Mapped Logical Channels (0...10)

unsigned char ucCrcChkResult (offset 0x60)

CRC check for FP Header and Payload:

0	no CRC error
1	Payload CRC error
2	Header CRC error

UMTS FP Iub/Iur Data Frame Trailer Section (V. 2)

usTrailerType=0x10 (not handled by reassembler)
or
usTrailerType=0x30 (handled by reassembler)
usVersion= 2

unsigned short usVPI (offset 0x8)

Virtual Path Identifier

unsigned short usVCI (offset 0xa)

Virtual Channel Identifier

unsigned char ucCID (offset 0xc)

Sub–Channel Identifier

unsigned char ucIfID (offset 0xd)

Interface Identifier

unsigned char ucRadioMode (offset 0xe)

Radio mode:

1	FDD
2	TDD 3.84 Mcps
3	TDD 1.28 Mcps

unsigned char ucDirection (offset 0xf)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned char ucInterface (offset 0x10)

UTRAN Interface type:

1	FP Iub
2	FP Iur

unsigned short usCFN (offset 0x14)

Connection Frame Number

unsigned char ucTrChType (offset 0x16)

Type of Transport Channel:

1	Broadcast Channel (BCH)
2	Paging Channel (PCH)
3	Common Packet Channel (CPCH)
4	Random Access Channel (RACH)
5	Forward Link Access Channel (FACH)
6	Uplink Shared Channel (USCH)
7	Downlink Shared Channel (DSCH)
8	Dedicated Channel (DCH)

unsigned char ucTFI (offset 0x17)

Transport Format: Index

unsigned long ulTrBlockSize (offset 0x18)

Transport Format: Transport Block Size (Bits)

unsigned long ulTrBlockSetSize (offset 0x1c)

Transport Format: Transport Block Set Size (Bits)

unsigned long ulTTI (offset 0x20)

Transport Format: Transmission Time Interval

unsigned char ucErrCodeType (offset 0x24)

Transport Format: Error Protection Code Type:

1	No Coding
2	Turbo Coding
3	Conventional Coding

unsigned char ucErrCodeRate (offset 0x25)

Transport Format: Error Protection Code Rate:

2	Half (1/2)
3	Third (1/3)

unsigned short usErrCodeMatching (offset 0x26)

Transport Format: Static Rate Matching Parameter

unsigned long ulCRCSize (offset 0x28)

Transport Format: Size of CRC

unsigned char ucLgChType

(10 entries: offsets 0x2c, 0x34, 0x3c, 0x44, 0x4c, 0x54, 0x5c, 0x64, 0x6c, 0x74)

Mapped Logical Channels (10 entries): Type of Logical Channel:

1	Broadcast Control Channel (BCCH)
2	Paging Control Channel (PCCH)
3	Dedicated Control Channel (DCCH)
4	Common Control Channel (CCCH)
5	Common Traffic Channel (CTCH)
6	Dedicated Traffic Channel (DTCH)
7	Shared Channel Control Channel (SHCCH)

unsigned char ucRlcMode

(10 entries: offsets 0x2d, 0x35, 0x3d, 0x45, 0x4d, 0x55, 0x5d, 0x65, 0x6d, 0x75)

Mapped Logical Channels (10 entries): RLC Mode:

1	Transparent Mode, segmented
2	Transparent Mode, non–segmented
3	Unacknowledged Mode
4	Acknowledge Mode

unsigned char ucLI

(10 entries: offsets 0x2e, 0x36, 0x3e, 0x46, 0x4e, 0x56, 0x5e, 0x66, 0x6e, 0x76)

Mapped Logical Channels (10 entries): Length of LI field (7 or 15)

unsigned char ucCTid

(10 entries: offset 0x2f, 0x37, 0x3f, 0x47, 0x4f, 0x57, 0x5f, 0x67, 0x6f, 0x77)

Mapped Logical Channels (10 entries): Identification of Logical Channel

(0...14; 255 for definite logical channel without CT id)

unsigned char ucPdcpType

(10 entries: offset 0x30, 0x38, 0x40, 0x48, 0x50, 0x58, 0x60, 0x68, 0x70, 0x78)

Mapped Logical Channels (10 entries): PDCP type

0	Unknown / RRC over RLC
1	Transparent
2	Non transparent

unsigned char ucNumLgChMapd (offset 0x7c)

Number of Mapped Logical Channels (0...10)

unsigned char ucCrcChkResult (offset 0x8c)

CRC check for FP Header and Payload:

0	no CRC error
1	Payload CRC error
2	Header CRC error

UMTS FP Iub/Iur Data Frame Trailer Section (V. 3)

usTrailerType=0x10 (not handled by reassembler)
or
usTrailerType=0x30 (handled by reassembler)
usVersion= 3

unsigned short usVPI (offset 0x8)

Virtual Path Identifier

unsigned short usVCI (offset 0xa)

Virtual Channel Identifier

unsigned char ucCID (offset 0xc)

Sub–Channel Identifier

unsigned char ucIfID (offset 0xd)

Interface Identifier

unsigned char ucRadioMode (offset 0xe)

Radio mode:

1	FDD
2	TDD 3.84 Mcps
3	TDD 1.28 Mcps

unsigned char ucDirection (offset 0xf)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned char ucInterface (offset 0x10)

UTRAN Interface type:

1	FP Iub
2	FP Iur

unsigned short usCFN (offset 0x14)

Connection Frame Number

unsigned char ucTrChType (offset 0x16)

Type of Transport Channel:

1	Broadcast Channel (BCH)
2	Paging Channel (PCH)
3	Common Packet Channel (CPCH)
4	Random Access Channel (RACH)
5	Forward Link Access Channel (FACH)
6	Uplink Shared Channel (USCH)
7	Downlink Shared Channel (DSCH)
8	Dedicated Channel (DCH)

unsigned char ucNumTrChMuxd (offset 0x17)

Number of Transport Channels (0 … 3)

unsigned char ucTFI

(3 entries: offsets 0x18, 0x80, 0xe8)

Transport Format: Index

unsigned char ucNumLgChMapd

(3 entries: offsets 0x19, 0x81, 0xe9)

Number of Mapped Logical Channels (0...10)

unsigned long ulTrBlockSize

(3 entries: offsets 0x1c, 0x84, 0xec)

Transport Format: Transport Block Size (Bits)

unsigned long ulTrBlockSetSize

(3 entries: offsets 0x20, 0x88, 0xf0)

Transport Format: Transport Block Set Size (Bits)

unsigned long ulTTI

(3 entries: offsets 0x24, 08c, 0xf4)

Transport Format: Transmission Time Interval

unsigned char ucErrCodeType

(3 entries: offsets 0x28, 0x90, 0xf8)

Transport Format: Error Protection Code Type:

1	No Coding
2	Turbo Coding
3	Conventional Coding

unsigned char ucErrCodeRate

(3 entries: offsets 0x29, 0x91, 0xf9)

Transport Format: Error Protection Code Rate:

2	Half (1/2)
3	Third (1/3)

unsigned short usErrCodeMatching

(3 entries: offsets 0x2a, 0x92, 0xfa)

Transport Format: Static Rate Matching Parameter

unsigned long ulCRCSize

(3 entries: offsets 0x2c, 0x94, 0xfc)

Transport Format: Size of CRC

unsigned char ucLgChType

(3 * 10 entries:)

(offsets 0x30, 0x38, 0x40, 0x48, 0x50, 0x58, 0x60, 0x68, 0x70, 0x78)

(offsets 0x98, 0xa0, 0xa8, 0xb0, 0xb8, 0xc0, 0xc8, 0xd0, 0xd8, 0xe0)

(offsets 0x100, 0x108, 0x110, 0x118, 0x120, 0x128, 0x130, 0x138, 0x140, 0x148)

Mapped Logical Channels (3 * 10 entries): Type of Logical Channel:

1	Broadcast Control Channel (BCCH)
2	Paging Control Channel (PCCH)
3	Dedicated Control Channel (DCCH)
4	Common Control Channel (CCCH)
5	Common Traffic Channel (CTCH)
6	Dedicated Traffic Channel (DTCH)
7	Shared Channel Control Channel (SHCCH)

unsigned char ucRlcMode

(3 * 10 entries:)

(offsets 0x31, 0x39, 0x41, 0x49, 0x51, 0x59, 0x61, 0x69, 0x71, 0x79)

(offsets 0x99, 0xa1, 0xa9, 0xb1, 0xb9, 0xc1, 0xc9, 0xd1, 0xd9, 0xe1)

(offsets 0x101, 0x109, 0x111, 0x119, 0x121, 0x129, 0x131, 0x139, 0x141, 0x149)

Mapped Logical Channels (3 * 10 entries): RLC Mode:

1	Transparent Mode, segmented
2	Transparent Mode, non–segmented
3	Unacknowledged Mode
4	Acknowledge Mode

unsigned char ucLI

(3 * 10 entries:)

(offsets 0x32, 0x3a, 0x42, 0x4a, 0x52, 0x5a, 0x62, 0x6a, 0x72, 0x7a)

(offsets 0x9a, 0xa2, 0xaa, 0xb2, 0xba, 0xc2, 0xca, 0xd2, 0xda, 0xe2)

(offsets 0x102, 0x10a, 0x112, 0x11a, 0x122, 0x12a, 0x132, 0x13a, 0x142, 0x14a)

Mapped Logical Channels (3 * 10 entries): Length of LI field (7 or 15)

unsigned char ucCTid

(3 * 10 entries:)

(offsets 0x33, 0x3b, 0x43, 0x4b, 0x53, 0x5b, 0x63, 0x6b, 0x73, 0x7b)

(offsets 0x9b, 0xa3, 0xab, 0xb3, 0xbb, 0xc3, 0xcb, 0xd3, 0xdb, 0xe3)

(offsets 0x103, 0x10b, 0x113, 0x11b, 0x123, 0x12b, 0x133, 0x13b, 0x143, 0x14b)

Mapped Logical Channels (3 * 10 entries): Identification of Logical Channel

(0...14; 255 for definite logical channel without CT id)

unsigned char ucPdcpType

(3 * 10 entries:)

(offsets 0x34, 0x3c, 0x44, 0x4c, 0x54, 0x5c, 0x64, 0x6c, 0x74, 0x7c)

(offsets 0x9c, 0xa4, 0xac, 0xb4, 0xbc, 0xc4, 0xcc, 0xd4, 0xdc, 0xe4)

(offsets 0x104, 0x10c, 0x114, 0x11c, 0x124, 0x12c, 0x134, 0x13c, 0x144, 0x14c)

Mapped Logical Channels (3 * 10 entries): PDCP type

0	Unknown / RRC over RLC
1	Transparent
2	Non transparent

unsigned char ucCrcChkResult (offset 0x17c)

CRC check for FP Header and Payload:

0	no CRC error
1	Payload CRC error
2	Header CRC error

UMTS FP Iub/Iur Control Frame Trailer Section (V. 0 or 1)

usTrailerType=0x11
usVersion= 0 or 1

unsigned short usVPI (offset 0x8)

Virtual Path Identifier

unsigned short usVCI (offset 0xa)

Virtual Channel Identifier

unsigned char ucCID (offset 0xc)

Sub–Channel Identifier

unsigned char ucIfID (offset 0xd)

Interface Identifier

unsigned char ucRadioMode (offset 0xe)

Radio mode:

1	FDD
2	TDD 3.84 Mcps
3	TDD 1.28 Mcps

unsigned char ucDirection (offset 0xf)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned char ucFpCtrlFrType (offset 0x10)

FP Iub Control Frame Type:

1	Outer Loop Power Control (DCH only)
2	Timing Adjustment
3	DL Synchronisation
4	UL Synchronisation
5	DL Signalling for DSCH (DCH only)
6	DL Node Synchronisation
7	UL Node Synchronisation
8	Rx Timing Deviation (DCH) / Dynamic PUSH Assignment (Common Channels)
9	Radio Interface Parameter Update (DCH) / Timing Advance (Common Channels)
10	Timing Advance (DCH only)

unsigned char ucTrChType (offset 0x11)

Type of Transport Channel:

1	Broadcast Channel (BCH)
2	Paging Channel (PCH)
3	Common Packet Channel (CPCH)
4	Random Access Channel (RACH)
5	Forward Link Access Channel (FACH)
6	Uplink Shared Channel (USCH)
7	Downlink Shared Channel (DSCH)
8	Dedicated Channel (DCH)

unsigned char ucCrcChkResult (offset 0x60)

CRC check for FP Header and Payload:

1	no CRC error
2	Payload CRC error
3	Header CRC error

UMTS FP Iub/Iur Control Frame Trailer Section (V. 2)

usTrailerType=0x11
usVersion= 2

unsigned short usVPI (offset 0x8)

Virtual Path Identifier

unsigned short usVCI (offset 0xa)

Virtual Channel Identifier

unsigned char ucCID (offset 0xc)

Sub–Channel Identifier

unsigned char ucIfID (offset 0xd)

Interface Identifier, Port number 0...31

unsigned char ucRadioMode (offset 0xe)

Radio mode:

1	FDD
2	TDD 3.84 Mcps
3	TDD 1.28 Mcps

unsigned char ucDirection (offset 0xf)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned char ucInterface (offset 0x10)

UTRAN Interface type:

1	FP Iub
2	FP Iur

unsigned char ucFpCtrlFrType (offset 0x14)

FP Iub Control Frame Type:

1	Outer Loop Power Control (DCH only)
2	Timing Adjustment
3	DL Synchronisation
4	UL Synchronisation
5	DL Signalling for DSCH (DCH only)
6	DL Node Synchronisation
7	UL Node Synchronisation
8	Rx Timing Deviation (DCH) / Dynamic PUSCH Assignment (Common Channels)
9	Radio Interface Parameter Update (DCH) / Timing Advance (Common Channels)
10	Timing Advance (DCH only)

unsigned char ucTrChType (offset 0x15)

Type of Transport Channel:

1	Broadcast Channel (BCH)
2	Paging Channel (PCH)
3	Common Packet Channel (CPCH)
4	Random Access Channel (RACH)
5	Forward Link Access Channel (FACH)
6	Uplink Shared Channel (USCH)
7	Downlink Shared Channel (DSCH)
8	Dedicated Channel (DCH)

unsigned char ucCrcChkResult (offset 0x8c)

CRC check for FP Header and Payload:

0	no CRC error
1	Payload CRC error
2	Header CRC error

UMTS FP Iub/Iur Control Frame Trailer Section (V. 3)

usTrailerType=0x11

usVersion= 3

unsigned short usVPI (offset 0x8)

Virtual Path Identifier

unsigned short usVCI (offset 0xa)

Virtual Channel Identifier

unsigned char ucCID (offset 0xc)

Sub–Channel Identifier

unsigned char ucIfID (offset 0xd)

Interface Identifier, Port number 0...31

unsigned char ucRadioMode (offset 0xe)

Radio mode:

1	FDD
2	TDD 3.84 Mcps
3	TDD 1.28 Mcps

unsigned char ucDirection (offset 0xf)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned char ucInterface (offset 0x10)

UTRAN Interface type:

1	FP Iub
2	FP Iur

unsigned char ucFpCtrlFrType (offset 0x14)

FP Iub Control Frame Type:

1	Outer Loop Power Control (DCH only)
2	Timing Adjustment
3	DL Synchronisation
4	UL Synchronisation
5	DL Signalling for DSCH (DCH only)
6	DL Node Synchronisation
7	UL Node Synchronisation
8	Rx Timing Deviation (DCH) / Dynamic PUSCH Assignment (Common Channels)
9	Radio Interface Parameter Update (DCH) / Timing Advance (Common Channels)
10	Timing Advance (DCH only)

unsigned char ucTrChType (offset 0x15)

Type of Transport Channel:

1	Broadcast Channel (BCH)
2	Paging Channel (PCH)
3	Common Packet Channel (CPCH)
4	Random Access Channel (RACH)
5	Forward Link Access Channel (FACH)
6	Uplink Shared Channel (USCH)
7	Downlink Shared Channel (DSCH)
8	Dedicated Channel (DCH)

unsigned char ucCrcChkResult (offset 0x17c)

CRC check for FP Header and Payload:

0	no CRC error
1	Payload CRC error
2	Header CRC error

UMTS FP Iub/Iur Reassembled Data Frame Trailer Section (V. 0 or 1)

usTrailerType=0x91
usVersion= 0 or 1

unsigned short usVPI (offset 0x8)

Virtual Path Identifier

unsigned short usVCI (offset 0xa)

Virtual Channel Identifier

unsigned char ucCID (offset 0xc)

Sub–Channel Identifier

unsigned char ucIfID (offset 0xd)

Interface Identifier, Port number 0...31

unsigned char ucRadioMode (offset 0xe)

Radio mode:

1	FDD
2	TDD 3.84 Mcps
3	TDD 1.28 Mcps

unsigned char ucDirection (offset 0xf)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned char ucTrChType (offset 0x10)

Type of Transport Channel:

1	Broadcast Channel (BCH)
2	Paging Channel (PCH)
3	Common Packet Channel (CPCH)
4	Random Access Channel (RACH)
5	Forward Link Access Channel (FACH)
6	Uplink Shared Channel (USCH)
7	Downlink Shared Channel (DSCH)
8	Dedicated Channel (DCH)

unsigned char ucDirection (offset 0x11)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned char ucRlcMode (offset 0x12)

RLC Mode:

1	Transparent Mode, segmented
2	Transparent Mode, non–segmented
3	Unacknowledged Mode
4	Acknowledge Mode

unsigned char ucLgChType (offset 0x13)

Type of Logical Channel:

1	Broadcast Control Channel (BCCH)
2	Paging Control Channel (PCCH)
3	Dedicated Control Channel (DCCH)
4	Common Control Channel (CCCH)
5	Common Traffic Channel (CTCH)
6	Dedicated Traffic Channel (DTCH)
7	Shared Channel Control Channel (SHCCH)

unsigned char ucCTid (offset 0x14)

Identification of Logical Channel

(0...14; 255 for definite logical channel without CT id)

unsigned char ucLayerAboveRlc (offset 0x15)

Layer above RLC (see RELATION in stack file also)

1	RRC_BCCH_FACH
2	RRC_BCCH_BCH
3	RRC_CCCH_DL
4	RRC_CCCH_UL
5	RRC_DCCH_DL
6	RRC_DCCH_UL
7	RRC_SHCCH_DL
8	RRC_SHCCH_UL
9	RRC_PCCH
21	USER DATA (PDCP transparent)
22	PDCP (non–transparent)

unsigned short usBitlen (offset 0x16)

Length of reassembled data in Bits

unsigned long ulError (offset 0x18)

Not used

unsigned char ucCrcChkResult (offset 0x60)

CRC check for FP Header and Payload:

0	no CRC error
1	Payload CRC error
2	Header CRC error

UMTS FP Iub/Iur Reassembled Data Frame Trailer Section (V. 2)

usTrailerType=0x92
usVersion= 2

unsigned short usVPI (offset 0x8)

Virtual Path Identifier

unsigned short usVCI (offset 0xa)

Virtual Channel Identifier

unsigned char ucCID (offset 0xc)

Sub–Channel Identifier

unsigned char ucIfID (offset 0xd)

Interface Identifier

unsigned char ucRadioMode (offset 0xe)

Radio mode:

1	FDD
2	TDD 3.84 Mcps
3	TDD 1.28 Mcps

unsigned char ucDirection (offset 0xf)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned char ucInterface (offset 0x10)

UTRAN Interface type:

1	FP Iub
2	FP Iur

unsigned char ucTrChType (offset 0x14)

Type of Transport Channel:

1	Broadcast Channel (BCH)
2	Paging Channel (PCH)
3	Common Packet Channel (CPCH)
4	Random Access Channel (RACH)
5	Forward Link Access Channel (FACH)
6	Uplink Shared Channel (USCH)
7	Downlink Shared Channel (DSCH)
8	Dedicated Channel (DCH)

unsigned char ucDirection (offset 0x15)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned char ucRlcMode (offset 0x16)

RLC Mode:

1	Transparent Mode, segmented
2	Transparent Mode, non–segmented
3	Unacknowledged Mode
4	Acknowledge Mode

unsigned char ucLgChType (offset 0x17)

Type of Logical Channel:

1	Broadcast Control Channel (BCCH)
2	Paging Control Channel (PCCH)
3	Dedicated Control Channel (DCCH)
4	Common Control Channel (CCCH)
5	Common Traffic Channel (CTCH)
6	Dedicated Traffic Channel (DTCH)
7	Shared Channel Control Channel (SHCCH)

unsigned char ucCTid (offset 0x18)

Identification of Logical Channel

(0...14; 255 for definite logical channel without CT id)

unsigned char ucLayerAboveRlc (offset 0x19)

Layer above RLC (see RELATION in stack file also)

1	RRC_BCCH_FACH
2	RRC_BCCH_BCH
3	RRC_CCCH_DL
4	RRC_CCCH_UL
5	RRC_DCCH_DL
6	RRC_DCCH_UL
7	RRC_SHCCH_DL
8	RRC_SHCCH_UL
9	RRC_PCCH
21	USER DATA (PDCP transparent)
22	PDCP (non–transparent)

unsigned short usBitlen (offset 0x1a)

Length of reassembled data in Bits

unsigned long ulError (offset 0x1c)

Not used

unsigned char ucCrcChkResult (offset 0x8c)

CRC check for FP Header and Payload:

0	no CRC error
1	Payload CRC error
2	Header CRC error

UMTS FP Iub/Iur Reassembled Data Frame Trailer Section (V. 3)

usTrailerType=0x92

usVersion= 3

unsigned short usVPI (offset 0x8)

Virtual Path Identifier

unsigned short usVCI (offset 0xa)

Virtual Channel Identifier

unsigned char ucCID (offset 0xc)

Sub–Channel Identifier

unsigned char ucIfID (offset 0xd)

Interface Identifier

unsigned char ucRadioMode (offset 0xe)

Radio mode:

1	FDD
2	TDD 3.84 Mcps
3	TDD 1.28 Mcps

unsigned char ucDirection (offset 0xf)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned char ucInterface (offset 0x10)

UTRAN Interface type:

1	FP Iub
2	FP Iur

unsigned char ucTrChType (offset 0x14)

Type of Transport Channel:

1	Broadcast Channel (BCH)
2	Paging Channel (PCH)
3	Common Packet Channel (CPCH)
4	Random Access Channel (RACH)
5	Forward Link Access Channel (FACH)
6	Uplink Shared Channel (USCH)
7	Downlink Shared Channel (DSCH)
8	Dedicated Channel (DCH)

unsigned char ucDirection (offset 0x15)

Port Direction:

0	Unknown
1	Uplink
2	Downlink

unsigned char ucRlcMode (offset 0x16)

RLC Mode:

1	Transparent Mode, segmented
2	Transparent Mode, non–segmented
3	Unacknowledged Mode
4	Acknowledge Mode

unsigned char ucLgChType (offset 0x17)

Type of Logical Channel:

1	Broadcast Control Channel (BCCH)
2	Paging Control Channel (PCCH)
3	Dedicated Control Channel (DCCH)
4	Common Control Channel (CCCH)
5	Common Traffic Channel (CTCH)
6	Dedicated Traffic Channel (DTCH)
7	Shared Channel Control Channel (SHCCH)

unsigned char ucCTid (offset 0x18)

Identification of Logical Channel

(0...14; 255 for definite logical channel without CT id)

unsigned char ucLayerAboveRlc (offset 0x19)

Layer above RLC (see RELATION in stack file also)

1	RRC_BCCH_FACH
2	RRC_BCCH_BCH
3	RRC_CCCH_DL
4	RRC_CCCH_UL
5	RRC_DCCH_DL
6	RRC_DCCH_UL
7	RRC_SHCCH_DL
8	RRC_SHCCH_UL
9	RRC_PCCH
21	USER DATA (PDCP transparent)
22	PDCP (non–transparent)

unsigned short usBitlen (offset 0x1a)

Length of reassembled data in Bits

unsigned long ulError (offset 0x1c)

Not used

unsigned char ucCrcChkResult (offset 0x17c)

CRC check for FP Header and Payload:

0	no CRC error
1	Payload CRC error
2	Header CRC error

UMTS FP Iu UP Data Frame Trailer Section (V. 1)

usTrailerType=0x50
usVersion= 1

unsigned short usVPI (offset 0x8)

Virtual Path Identifier

unsigned short usVCI (offset 0xa)

Virtual Channel Identifier

unsigned char ucCID (offset 0xc)

Sub–Channel Identifier

unsigned char ucIfID (offset 0xd)

Interface Identifier, Port number 0...31

unsigned char ucCrcChkResult (offset 0xe)

CRC check for FP Header and Payload:

0	no CRC error
1	Payload CRC error
2	Header CRC error

Using the DLL

Loading the Library

To use the library you have to load the DLL. There are two ways to do this:

- Include the *K12RecFl.lib*, this is the easiest way to use the functions under Visual C++ (see section *Loading of the DLL by Using K12RecFl.lib*)
- Import the DLL directly. This capability is available in most programming languages (programming manual of your compiler). You have to load all the needed functions of the DLL manually (see section *Importing of Needed Functions*).

Loading of the DLL by Using K12RecFl.lib

The easiest way to use a DLL under Visual C++ is to link the *K12RecFl.lib* with your project. To do so, proceed as follows:

1. Copy the *K12RecFl.lib* into your project directory (or into a standard library directory).

2. Add the library to your link list (Project–>Settings–>Linker–>object/library–module).

Now you can use the all commands of the DLL in your own program.

```
...
#include ”K12RecFt.h”
...
int iRecFilehandle = K12_REC_OpenRecFileRead (”TestFile.rf5”, OPEN_FILE);    // Opens the Record file
K12_REC_stEventHead_t * pstEventHead;
long plEvNo, lEventLen;
pstEventHead = K12_REC_GetFirst (iRecFilehandle, &plEvNo, &lEventLen);  // Gets the first event
...
```

Importing of Needed Functions

With more effort you can load each used function manually. Similar to this you can also use the DLL from other programming languages.

The following code fragment shows an example of how to load the DLL under C and C++:

```
...
#include ”K12RecFl.h”
...
HINSTANCE hLibInstance = LoadLibrary(”K12RecFl”);            // Open the Library
...
K12_REC_PROC_OPEN_RECFILE_READ OpenRecFileRead;
OpenRecFileRead = (K12_REC_PROC_OPEN_RECFILE_READ)GetProcAddress(hLibIns-
tance,”K12_REC_OpenRecFileRead”);
K12_REC_PROC_GET_FIRST GetFirst;
GetFirst = (K12_REC_PROC_GET_FIRST)GetProcAddress(hLibInstance,”K12_REC_GetFirst”);
...
int iRecFilehandle = K12_REC_OpenRecFileRead (”TestFile.rf5”, OPEN_FILE);    // Opens the record file
K12_REC_stEventHead_t * pstEventHead;
long plEvNo, lEventLen;
pstEventHead = GetFirst (iRecFilehandle, &plEvNo, &lEventLen);  // Gets the first event
...
FreeLibrary(hLibInstance);   // Close the library
...
```

1. You have to include the *K12RecFl.h* first, which defines all needed parameters and prototypes (for example the definition of K12_REC_PROC_OPEN_RECFILE_READ).

2. The next step is to load the DLL.

   HINSTANCE hLibInstance = LoadLibrary(”K12RecFl”); // Open the Library

3. Then you have to create a pointer to each used filetool function:

   K12_REC_PROC_GET_FIRST GetFirst;

   GetFirst = (K12_REC_PROC_GET_FIRST)GetProcAddress(hLibInstance,”K12_REC_GetFirst”);

The prototypes of all usable functions are defined in the header file *K12RecFl.h*. Use the prototype name defined in the command description (*Function Prototype*).

4. Then, you can use the function normaly:

   pstEventHead = GetFirst (iRecFilehandle, &plEvNo, &plEvLen);

5. At the end of your application you should close the DLL:

   FreeLibrary(hLibInstance);

Handling Events

The following sample code fragment is an extract of an program that gets the next event and looks after an LDS (logical data source) or a frame event. If it matches, it coverts the pointer to the event header structure into the event specific structure. A complete sample is shown in the demo file *ListEvnt.c*

```
...
K12_REC_stEventHead_t * pstEventHead;
long plEvNo, lEventLen;
...
pstEventHead = pGetNextFunc (hRecFilehandle, &plEvNo, &lEventLen);  // Gets the next Event
if (pstEventHead)
{
   switch(pstEventHead–>usGroup)
   {
   case K12_REC_EVENTGRP_RFCONF:   // it’s a config. Event
        if(pstEventHead–>usType == K12_REC_EVENTTYP_LDS_CONF)
        {   // Event is an LDS–Config Event –> cast Pointer to Event–Header to LDS–Event–structure
            K12_REC_stEventLDSConfig2_t * pstLDSEvent;
            pstLDSEvent = (K12_REC_stEventLDSConfig2_t *) pstEventHead;
            ...
        }
     break;

   case K12_REC_EVENTGRP_DATA:   // it’s a data. Event
        if(pstEventHead–>usType == K12_REC_EVENTTYP_FRAME)
        {   // Event is an Frame Event –> cast Pointer to Event–Header to Event–Frame–structure
            K12_REC_stEventFrame_t * pstFrameEvent;
            pstFrameEvent = (K12_REC_stEventFrame_t *) pstEventHead;
            ...
        }
     break;
   }
   ...
}
...
```

Command Set

In the following command description, the values of the parameters or results of some commands are defined in the header file *K12RecFl.h*. If you cannot use the header file – for example because you do not use C or C++ as programming languages – use the value that follows in parentheses.

Commands to Read Record Files

Control Commands

int K12_REC_OpenRecFileRead (
char * pczFilename,
long lFlags)

Description

Opens a file to read.

Input Parameters

pczFilename
Name of record file (including path, filename and file extension).

lFlags
For future use (should be set to 0L).

Return Value

Handle
When opening was successful (has to be stored, because the handle is used for any other handling with the file).

0
If it wasn’t possible to open the file.

Function Prototyp

K12_REC_PROC_OPEN_RECFILE_READ

Example

int iReadFileHandle =
K12_REC_OpenRecFileRead(”c:\\Recordfiles\\DemoRec.rf5”, 0L);

See also:
K12_REC_CloseRecFile

K12_REC_RetVal_t K12_REC_SetTimestampCalcMode (int ihandle , K12_REC_TimestampMode_t TimestampMode)

Description

Determinates whether a timestamp from a non K1205 type record file should be translated (calculated).

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

TimestampMode

- K12_REC_DO_NOT_CALC (0)
  Do not calculate timestamps (increases speed). If the record file is not a K1205 type, the timestamps in the returned events are set to zero (**default**).
- K12_REC_DO_CALC (1)
  The timestamp will be converted to the K1205 format.

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototyp

K12_REC_PROC_SET_TIMESTAMP_MODE

Example

```
K12_REC_SetTimestampCalcMode(iReadFileHandle, K12_REC_Do_Calc);
 // activate timestamp–  converting
```

See also:
K12_REC_OpenRecFileRead,
K12_REC_GetTimestampCalcMode,
K12_REC_GetTimestampFormat

Status Commands

K12_REC_RetVal_t K12_REC_ShouldSwap (
int ihandle,
int *piResult)

Description

If the byte order of the record file is not the same as the byte order of the local machine, the data must be swapped to use them. This function indicates whether the record file has the same byte order as the local CPU.

Input Parameters

ihandle: Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

piResult:
Address of a long that returns the result.

- 0
  Same byte order (don’t need to swap).
- 1
  Different byte order (need to swap).

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototyp

K12_REC_PROC_SHOULD_SWAP

Example

```
int iResult;
K12_REC_RetVal_t Ret = K12_REC_ShouldSwap(iReadFileHandle, &iResult);
if (Ret == K12_REC_OK)
  printf(”Result: %d”, iResult);
```

See also:
K12_REC_OpenRecFileRead,
K12_REC_GetSwapMode,
K12_REC_SetSwapMode

K12_REC_RetVal_t K12_REC_GetFileType (
int ihandle ,
K12_REC_FileType_t * pFileType)

Description

Gets the type of the record file opened.

Input Parameters

ihandle: Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

pFileType
Address of a K12_REC_FileType that returns the type of file.

- K12_REC_K1205TYP (1)
  a K15, K1205, K1297-G20 record file is open.
- K12_REC_K1103TYP (2)
  a K1103 record file is open.
- K12_REC_K1297TYP (5)
  a K1297-Classic record file is open.

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototyp

K12_REC_PROC_GET_FILE_TYPE

Example

```
K12_REC_FileType_t FileType ;
K12_REC_RetVal_t Ret = K12_REC_GetCaptureType(iReadFileHandle,
&FileType);
if ((Ret == K12_REC_OK) && (FileType == K12_REC_K1205TYP))
  printf(”K1205 file”);
```

See also:
K12_REC_OpenRecFileRead

K12_REC_RetVal_t K12_REC_GetTimestampCalcMode (
int ihandle
K12_REC_TimestampMode_t * pTimestampMode)

Description

Returns whether a timestamp should be translated (calculated).

The translation–mode can be changed by K12_REC_SetCalcTimestamp(...) (see *Control Commands* in section *Commands to Read Record Files*).

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

pTimestampMode
Address of an K12_REC_TimestampMode_t that returns the timestamp mode.

- K12_REC_DO_NOT_CALC (0)
  Don’t calculate timestamps. If the record file isn’t a K1205 type, the timestamps in the returned events are set to Zero.
- K12_REC_DO_CALC (1)
  If the record file is not a K1205 type, it has another timestamp format. The timestamp will be converted to the K1205 format.

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototyp

K12_REC_PROC_GET_TIMESTAMP_MODE

Example

K12_REC_TimestampMode_t TimestampMode;
K12_REC_RetVal_t Ret = K12_REC_GetTimestampCalcMode(iReadFileHandle, &TimestampMode);

See also:
K12_REC_OpenRecFileRead,
K12_REC_SetTimestampCalcMode,
K12_REC_GetTimestampFormat

K12_REC_RetVal_t K12_REC_GetTimestampFormat (
int ihandle,
long * plSecsOfBase,
long * plTicksPerSec)

Description

Returns information about the timestamp format that was used to create the record file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

plSecsOfBase
Address of a long that returns the Seconds of Base.

plTicksPerSec
Address of a long that returns the Ticks per second.

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototyp

K12_REC_PROC_GET_TIMESTAMP_FORMAT

Example

```
long lSecsOfBase, lTicksPerSec;
if(K12_REC_GetTimestampFormat (iReadFileHandle, &SecsOfBase,
&TicksPerSec))
   printf (”Seconds of Base: %ld  ; Ticks per second: %ld”, SecsOfBase,
   TicksPerSec);
```

See also:
K12_REC_OpenRecFileRead,
K12_REC_SetTimestampCalcMode,
K12_REC_GetTimestampCalcMode

***Note.** In most cases the 64 bit time stamp is a 64 bit value that represents 'X' ticks per second since absolute date 'Y'. The 'seconds of base' are the seconds that have past since 01.01.1970 00:00:00. When you load K15, K1297-G20, K1205, or K1103 record files, the standard base is 631.152.000 (representing 01.01.1990 00:00:00). For K1297-Classic recordings normally the base is 788.918.400 (01.01.1995 00:00:00).*

The returned value of 'Ticks per second' depends on the timer of the measurement system: The reciprocal of this value is the timer base of the system (in seconds).

In both returned values are zero, another kind of timestamp format is indicated. This format has two 32 bit values instead of one 64 bit value. The upper 32 bit represent the seconds since 01.01.1970 00:00:00 (so you can use all standard C time() functions on it), and the lower 32 bits represent the nanoseconds inside this second. This format (called general timestamp format*) is not currently used but may be used in the future.*

K12_REC_RetVal_t K12_REC_GetEventPosInPercent (int ihandle, int * piPosition)

Description

Gives a rough position of the last read event relative to all existing events in the record file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

piPosition
Address of an integer that returns the position in the file (as a percent).

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototyp

K12_REC_PROC_GET_EVENT_POS_IN_PERCENT

Example

```
int iPosition ;
K12_REC_RetVal_t Ret = K12_REC_GetEventPosInPercent(iReadFileHandle,
&iPosition);
if (Ret == K12_REC_OK)
   printf(”Position is at %d \%”, iPosition);
```

See also:
K12_REC_OpenRecFileRead,
K12_REC_GetFirst

Reading Commands

const K12_REC_stEventHead_t * K12_REC_GetFirst (
int ihandle,
long * plEventNo,
long * plEventLen)

Description

Gets the first event in the record file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

plEventNo
Address of a long that returns the number of the event.

plEventLen
Address of a long that returns the length of the event.

Return Value

<> NULL
Pointer to an eventhead structure (see section *Event Header*)

NULL
No valid event exists.

Function Prototyp

K12_REC_PROC_GET_FIRST

Example

```
long lEventNo, lEventLen;
K12_REC_stEventHead_t * pstEventHead;
pstEventHead = K12_REC_GetFirst(iReadFileHandle, &lEventNo,
& lEventLen);
if(!pstEventHead)
   return;  // no event present
```

See also:
K12_REC_OpenRecFileRead, K12_REC_GetLast,
K12_REC_GetPrev, K12_REC_GetNext,
K12_REC_GetThis, K12_REC_GetFirstConf,
K12_REC_GetNextConf

const K12_REC_stEventHead_t * K12_REC_GetLast (int ihandle, long * plEventNo, long * plEventLen)

Description

Gets the last event in the record file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

plEventNo
Address of a long that returns the number of the event.

plEventLen
Address of a long that returns the length of the event.

Return Value

<> NULL
Pointer to an eventhead structure (see section *Event Header*).

NULL
No valid event exists.

Function Prototyp

K12_REC_PROC_GET_LAST

Example

```
long lEventNo, lEventLen;
K12_REC_stEventHead_t * pstEventHead;
pstEventHead = K12_REC_GetLast(iReadFileHandle, &lEventNo,
& lEventLen);
if(!pstEventHead)
   return;  // no event present
```

See also:
K12_REC_OpenRecFileRead, K12_REC_GetFirst,
K12_REC_GetPrev, K12_REC_GetNext,
K12_REC_GetThis, K12_REC_GetFirstConf,
K12_REC_GetNextConf

const K12_REC_stEventHead_t * K12_REC_GetPrev (int ihandle, long * plEventNo, long * plEventLen)

Description

Gets the previous event and decrements the readout position in the record file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

plEventNo
Address of a long that returns the number of the event.

plEventLen
Address of a long that returns the length of the event.

Return Value

<> NULL
Pointer to an eventhead structure (see section *Event Header*).

NULL
No valid event exists.

Function Prototyp

K12_REC_PROC_GET_PREV

Example

```
long lEventNo, lEventLen;
K12_REC_stEventHead_t * pstEventHead;
pstEventHead = K12_REC_GetPrev(iReadFileHandle, &lEventNo,
& lEventLen);
if(!pstEventHead)
   return;  // no event present
```

See also:
K12_REC_OpenRecFileRead, K12_REC_GetFirst,
K12_REC_GetLast, K12_REC_GetNext,
K12_REC_GetThis, K12_REC_GetFirstConf,
K12_REC_GetNextConf

const K12_REC_stEventHead_t * K12_REC_GetNext (
int ihandle,
long * plEventNo
long * plEventLen)

Description

Gets the next event and increments the readout position in the record file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

plEventNo
Address of a long that returns the number of the event.

plEventLen
Address of a long that returns the length of the event.

Return Value

<> NULL
Pointer to an eventhead structure (see section *Event Header*).

NULL
No valid event exists.

Function Prototyp

K12_REC_PROC_GET_NEXT

Example

```
long lEventNo, lEventLen;
K12_REC_stEventHead_t * pstEventHead;
pstEventHead = K12_REC_GetNext(iReadFileHandle, &lEventNo,
& lEventLen);
if(!pstEventHead)
   return;  // no event present
```

See also:
K12_REC_OpenRecFileRead, K12_REC_GetFirst,
K12_REC_GetLast, K12_REC_GetPrev,
K12_REC_GetThis, K12_REC_GetFirstConf,
K12_REC_GetNextConf

const K12_REC_stEventHead_t * K12_REC_GetThis (
int ihandle,
long lWantedEventNo,
long * plEventNo,
long * plEventLen)

Description

Gets an specific event in the record file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

lWantedEventNo
Position of the wanted event (number of the event).

Output Parameters

plEventNo
Address of a long that returns the number of the event.

plEventLen
Address of a long that returns the length of the event.

Return Value

<> NULL
Pointer to an eventhead structure (see section *Event Header*).

NULL
No valid event exists.

Function Prototyp

K12_REC_PROC_GET_THIS

Example

```
long lEventNo, lEventLen;
K12_REC_stEventHead_t * pstEventHead;
pstEventHead = K12_REC_GetThis(iReadFileHandle, 100, &lEventNo,
& lEventLen);
// Try to get the entry no. 100
if(!pstEventHead)
   return;  // no event present
```

See also:
K12_REC_OpenRecFileRead, K12_REC_GetFirst,
K12_REC_GetLast, K12_REC_GetPrev,
K12_REC_GetNext, K12_REC_GetFirstConf,
K12_REC_GetNextConf

const K12_REC_stEventHead_t * K12_REC_GetFirstConf (int ihandle, long * plEventNo, long * plEventLen)

Description

An interface to get only the configuration events. Needed to get LDS (logical data source) configuration and LL configuration events without scanning the whole record file and without knowing the details of this record file format. Gets the first configuration event in the record file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

plEventNo
Address of a long that returns the number of the event.

plEventLen
Address of a long that returns the length of the event.

Return Value

<> NULL
Pointer to an eventhead structure (see section *Event Header*).

NULL
No valid event exists.

Function Prototyp

K12_REC_PROC_GET_FIRST_CONF

Example

```
long lEventNo, lEventLen;
K12_REC_stEventHead_t * pstEventHead;
pstEventHead = K12_REC_GetFirstConf(iReadFileHandle, &lEventNo,
& lEventLen);
if(!pstEventHead)
   return;  // no event present
```

See also:
K12_REC_OpenRecFileRead, K12_REC_GetFirst,
K12_REC_GetLast, K12_REC_GetPrev,
K12_REC_GetNext, K12_REC_GetThis,
K12_REC_GetNextConf, K12_REC_GetNumConfigEvents

const K12_REC_stEventHead_t * K12_REC_GetNextConf (
int ihandle,
long * plEventNo,
long * plEventLen)

Description

An interface to get only the configuration events. Needed to get LDS (logical data source) configuration and LL configuration events without scanning the whole record file and without knowing the details of this record file format. Gets the next configuration event in the record file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

plEventNo
Address of a long that returns the number of the event.

plEventLen
Address of a long that returns the length of the event.

Return Value

<> NULL
Pointer to an eventhead structure (see section *Event Header*).

NULL
No valid event exists.

Function Prototyp

K12_REC_PROC_GET_NEXT_CONF

Example

```
long lEventNo, lEventLen;
K12_REC_stEventHead_t * pstEventHead;
pstEventHead = K12_REC_GetNextConf(iReadFileHandle, &lEventNo,
& lEventLen);
if(!pstEventHead)
   return;  // no event present
```

See also:
K12_REC_OpenRecFileRead, K12_REC_GetFirst,
K12_REC_GetLast, K12_REC_GetPrev,
K12_REC_GetNext, K12_REC_GetThis,
K12_REC_GetFirstConf, K12_REC_GetNumConfigEvents

Note. *To use this command, first place the 'event–pointer' at the beginning of the configuration events (use K12_REC_GetFirstConf(...) see section Reading Commands).*

Commands to Write Record Files

Control Commands

```
int                 K12_REC_OpenRecFileWrite (
char * pczFilename,
K12_REC_WMode_t Writemode,
long lFilesize,
K12_REC_OMode_t Openmode)
```

Description

Opens a file for writing.

Input Parameters

pczFilename
Name of record file(including path, filename and file extension).

Writemode:

- K12_REC_WMODE_VAR_SIZE (8)
  Files have a variable size (minimum 1 block, maximum disk full) and grow with the number of written blocks (endless).
- K12_REC_WMODE_MAX_SIZE (16)
  No write possible when maximum file size is reached (file size grows like K12_REC_WMODE_VAR_SIZE). The given file size (Parameter ‘Filesize’) is rounded down to the next full block.
- K12_REC_WMODE_FIX_SIZE (48)
  Like K12_REC_WMODE_MAX_SIZE, but the file size is specified (file size cannot grow).

lFilesize
Maximum size of the written File. Only needed for ‘Writemode’ K12_REC_WMODE_MAX_SIZE and K12_REC_WMODE_FIX_SIZE.

Openmode

- K12_REC_OMODE_OVERWRITE (50)
  Create a new file or overwrite an existing file.
- K12_REC_OMODE_APPEND (51)
  Continue existing file or create a new one.

***CAUTION.** Appended files must have the same file properties as the opened file.*

Return Value

> 0
Handle to opened file (is used for any other handling with the file).

< 0
Error

Function Prototyp

K12_REC_PROC_OPEN_RECFILE_WRITE

Example

```
int iWritehandle = K12_REC_OpenRecFileWrite(
”test.rf5”,K12_REC_ WMODE_VAR_SIZE, 0L,
K12_REC_OMODE_APPEND);
if(iWritehandle < 0)
{
  const char * pczErrString = K12_REC_GetErrorString(iWritehandle);
  printf(”File open error: %s”, pczErrString);
  return;          // Can’t open file
}
```

See also:
K12_REC_CloseRecFile,
K12_REC_GetErrorString

Possible Error Codes

ERRWR_WRONG_OMODE (–1)
Wrong open–mode

ERRWR_WRONG_WMODE (–2)
Wrong write–mode; can also happen when continued. File was created with a different write–mode than the current one

ERRWR_CANT_OPEN_FILE (–3)
No filename or wrong filename–length

ERRWR_IO_ERROR (–4)
Error in open(), lseek(), write() or read()

ERRWR_ISOPEN (–5)
Record file is already open

ERRWR_NO_MEMORY (–6)
No memory available

ERRWR_MUST_REP (–7)
Record file must be repaired

ERRWR_WRONG_OLD_FILE (–8)
File that should be appended is not a K15, a K1297-G20, or a K1205 record file

WRBL_IO_FAILED (–10)
lseek() or write() failed

WRBL_BAD_READ (–11)
Read error

WRBL_CANT_OPEN (–12)
No filename or wrong filename–length

WRBL_BAD_SIZE (–13)
Wrong file size (file too big)

WRBL_NO_FD (–14)
It was not possible to create a duplicate of the file handle

Status Commands

const char * K12_REC_GetErrorString (int iErrorNumber)

Description

Returns the English plain text of the error of the K12_REC_OpenRecFileWrite function.

Input Parameters

iErrorNumber
error number (< 0) returned by K12_REC_OpenRecFileWrite (see *Control Commands* in section *Commands to Write Record Files*).

Return Value

Pointer to English error string.

Function Prototyp

K12_REC_PROC_GET_ERROR_STRING

Example

```
int iWritehandle = K12_REC_OpenRecFileWrite(”test.rf5”,
WMODE_VAR_SIZE, 0L,   OMODE_OVERWRITE);

if(iWritehandle < 0)
{
  const char * pczErrString = K12_REC_GetErrorString(iWritehandle);
  printf(”File open Error: %s”, pczErrString);
  return; // Can’t open file
}
```

See also:
K12_REC_OpenRecFileWrite

Writing Commands

K12_REC_RetVal_t K12_REC_Write (
int ihandle,
K12_REC_stEventHead_t * pstOrgEvent)

Description

Write one event to the open file. All written events are buffered in a block, and only full blocks are written to the disk. After a write error, the file has to be closed normally.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileWrite see *Control Commands* in section *Commands to Write Record Files*).

Output Parameters

pstOrgEvent
Pointer to event, which should be written to file.

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error (no write possible, because file size – see *Control Commands* in section *Commands to Write Record Files* – is reached or disk is full; event is not saved!

Function Prototyp

K12_REC_PROC_WRITE

Example

```
long lEventNo, lEventLen;

K12_REC_stEventHead_t * pEventheader =
K12_REC_GetFirst(iReadFileHandle, &lEventNo, &lEventLen);

// Read Event from open Record file

if(!pEventheader)
   return;  // no event present
if(!iK12_REC_Write(iWritehandle, pEventheader))
   printf(”Can’t write event to file”);
```

See also:
K12_REC_OpenRecFileWrite,
K12_REC_CloseRecFile,
K12_REC_Flush

Note. *All data in a record file should be in Motorola format (MSBF). The Write() function only swaps (if necessary) the values of the event header ('K12_REC_stEventHead_t') and not the rest of an event!*

K12_REC_RetVal_t K12_REC_Flush (
int ihandle,
K12_REC_FlushFlag_t Flags)

Description

Flushes the current data to file. The file header does not contain the current data.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileWrite see *Control Commands* in section *Commands to Write Record Files*).

Flags:

- K12_REC_FLUSH_QUICK (0)
  writes all 'full' blocks to file (quick).
- K12_REC_FLUSH_ALL (1)
  writes all 'full' blocks plus the current (part filled) block including the last written events.

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototype

K12_REC_PROC_FLUSH

Example

```
if(iK12_REC_Write(iWritehandle, pEventheader) == K12_REC_ERROR)
{
  printf("Can't write event to file");
  K12_REC_Flush(iWritehandle, K12_REC_FLUSH_ALL);  // Flush block
  K12_REC_CloseRecFile(iWritehandle);
  return;
}
```

See also:
K12_REC_OpenRecFileWrite,
K12_REC_CloseRecFile,
K12_REC_Write

Commands to Read and Write Open Record Files

In the following all commands are listed, which can be used to read open files and to write open files.

Control Commands

K12_REC_RetVal_t K12_REC_CloseRecFile (int ihandle)

Description

Closes the open file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototype

K12_REC_PROC_CLOSE_RECFILE

Example

```
K12_REC_CloseRecFile(iReadFileHandle);
// Closes the file
```

See also:
K12_REC_OpenRecFileRead,
K12_REC_OpenRecFileWrite

K12_REC_RetVal_t K12_REC_SetSwapMode (int ihandle, K12_REC_SwapMode_t SwapMode)

Description

Changes the swap mode.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

iSwapMode:

- NO_SWAP (0)
  Do not swap any structures.
- AUTO_SWAP (1)
  Swaps any known structures if it's needed (**default**).
- DO_SWAP (2)
  Swap always. May be used, if the read data must be sent to another machine with the opposite byte order.

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototype

K12_REC_PROC_SET_SWAP_MODE

Example

```
K12_REC_SetSwapMode(iReadFileHandle, NO_SWAP);
// disables swapping
```

See also:
K12_REC_ShouldSwap,
K12_REC_GetSwapMode

Note. *If this function swaps data and the record file contains unknown structures, the Get...() functions return an error!*

Status Commands

const char * K12_REC_GetFileName (int ihandle)

Description

Gets the name of an open record file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Return Value

Pointer to the filename string.

Function Prototype

K12_REC_PROC_GET_FILENAME

Example

```
const char * pczFilename = K12_REC_GetFileName(iReadFileHandle);
printf(”Name of the open file: %s”, pczFilename);
```

See also:
K12_REC_OpenRecFileRead,
K12_REC_OpenRecFileWrite

K12_REC_RetVal_t K12_REC_GetSwapMode (
int ihandle,
K12_REC_SwapMode_t * pSwapMode)

Description

Gets Information about the current swap mode. The mode can be changed by the K12_REC_SetSwapMode(...) command.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileRead see *Control Commands* in section *Commands to Read Record Files*).

Output Parameters

pSwapMode
Address of an int that returns the swap mode.

- NO_SWAP (0)
  write events as they come.
- AUTO_SWAP (1)
  swap event if needed (default).
- DO_SWAP (2)
  always swap events.

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototype

K12_REC_PROC_GET_SWAP_MODE

Example

```
K12_REC_SwapMode_t CurrentSwapMode;
K12_REC_RetVal_t Ret= K12_REC_GetSwapMode(iReadFileHandle,
&CurrentSwapMode);
if (Ret == K12_REC_OK)
   printf(”Swap mode:: %d”, iCurrentSwapMode);
```

See also:
K12_REC_ShouldSwap,
K12_REC_SetSwapMode

Note. *Swapping means changing of the byte-order of the recording. This is necessary because the measurement boards of the K1205 use a Motorola processor (Motorola byte order format: MSBF – most significant byte first). In contrast, most PCs use a Intel processor (Intel byte order format: LSBF – least significant byte first). K1205 record files are stored in Motorola format. If you want to handle the record on a PC, record files have to be swapped.*

K12_REC_RetVal_t K12_REC_GetNumConfigEvents (
int ihandle,
int * piResult)

Description

Gets the number of configuration events in the open record file.

Input Parameters

ihandle
Handle to an open file (returned from K12_REC_OpenRecFileWrite see *Control Commands* in section *Commands to Write Record Files*).

Output Parameters

piResult
Address of an integer that returns the number of configuration events.

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototype

K12_REC_PROC_GET_NUM_CONFIG_EVENTS

Example

```
int iNumOfConfigEvents;
K12_REC_RetVal_t Ret = K12_REC_GetNumConfigEvents (iHandle, &iNumOfConfigEvents);
if(Ret == K12_REC_OK)
   printf(”File contents %ld Configuration events”, iNumOfConfigEvents);
```

See also:
K12_REC_OpenRecFileRead,
K12_REC_OpenRecFileWrite

Independent Status Commands

In the following all commands are listed, which are independent of an open file.

File Information Commands

K12_REC_RetVal_t K12_REC_GetOpenModes (
const char * pczFilename,
K12_REC_WMode_t * pWritemode,
long * plFilesize,
K12_REC_OMode_t * pOpenmode)

Description

Returns the parameters this record file was created with.

Input Parameters

pczFilename
Name of record file(including path, filename and file extension).

Output Parameters

pWritemode
Pointer to an K12_REC_WMode_t that returns the Writemode (if this value is not needed, set it to NULL).

plFilesize
Pointer to an long that returns the Filesize (if this value is not needed, set it to NULL).

pOpenmode
Pointer to an K12_REC_OMode_t that returns the Openmode (if this value is not needed, set it to NULL).

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototype

K12_REC_PROC_GET_OPEN_MODES

Example

```
long lFilesize;
K12_REC_OMode_t OpenMode;
K12_REC_WMode_t WriteMode;
K12_REC_RetVal_t iRet = K12_REC_GetOpenModes(”test.rf5”, &WriteMode,
&lFilesize, &OpenMode);
if(iRet == K12_REC_OK)
   printf(”Filesize: %ld ”, lFilesize);
```

See also:
K12_REC_OpenRecFileWrite

Commands To Convert Timestamps

```
K12_REC_RetVal_t                K12_REC_ConvertTimestampToGenericTime(

unsigned long    ulTsHigh,
unsigned long    ulTsLow,
unsigned long *  pulTimeInSec,
unsigned long *  pulNanoSec)
```

Description

Converts a timestamp into generic time.

Input Parameters

ulTsHigh
Bits 32 to 63 of the timestamp.

ulTsLow
Bits 0 to 31 of the timestamp.

Output Parameters

pulTimeInSec
Address of a long that returns the seconds since 1.1.1970 0:00:00

pulNanoSec
Address of a long that returns the Nanoseconds inside ulTimeInSec

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototype

K12_REC_PROC_CONVERT_TS_TO_GENERIC_TIME

Example

```
#include <time.h>
....
K12_REC_RetVal_t Ret;
K12_REC_stEventHead_t * pstEventHead;
if(!iRecFilehandle)
  return 0;
Ret = K12_REC_SetTimestampCalcMode(iRecFile-
handle,K12_REC_DO_CALC);
if(!Ret)
  return 0;

pstEventHead = K12_REC_GetThis (iRecFilehandle, 100, &plEvNo,
&lEventLen);  // Gets event 100
if(pstEventHead)
{
  if((pstEventHead–>usGroup == K12_REC_EVENTGRP_DATA) &&
  (pstEventHead–>usType == K12_REC_EVENTTYP_FRAME))
  {
    K12_RECstEventFrame_t * pstEventFrame = (K12_RECstEvent–Frame_t
    *) pstEventHead;
    unsigned long ulTimeInSec, ulNanoSec;
    Ret = K12_REC_ConvertTimestampToGenericTime( pstEvent–Frame–>
    ulTimestampHigh, pstEventFrame–> ulTimestampHigh, &ulTimeInSec,
    &ulNanoSec);
    if(!Ret)
      return;
    struct tm *newtime;
    newtime = gmtime((time_t *) &ulTimeInSec );  /* Convert time to struct tm
    form */
    printf( ”date and time of the frame timestamp are: %s”, asctime(newtime ) );
  }
}  /* if(pstEventHead) */
```

See also:
K12_REC_ConvertGenericTimeToTimestamp

Note. *Generic Time is the time a PC works with. The base of this is the 1/1/1970 00:00:00. When you convert timestamps in the generic time, you have the possibility to use System functions as ‘gmtime’, ‘asctime’ and so on (defined in ‘time.h’) to process the timestamps.*

K12_REC_RetVal_t K12_REC_ConvertGenericTimeToTimestamp(
unsigned long ulTimeInSec,
unsigned long ulNanoSec,
unsigned long * pulTsHigh,
unsigned long * pulTsLow)

Description

Converts generic time into a timestamp.

Input Parameters

ulTimeInSec
Seconds since 1.1.1970 0:00:00

ulNanoSec:
Nanoseconds inside ulTimeInSec

Outut Parameters

pulTsHigh
Address of a long that returns Bits 32 to 63 of the timestamp.

pulTsLow
Address of a long that returns the Bits 0 to 31 of the timestamp.

Return Value

K12_REC_OK (1)
Ok

K12_REC_ERROR (0)
Error

Function Prototype

K12_REC_PROC_CONVERT_GENERIC_TIME_TO_TS

Example

```
#include <time.h>
....
K12_REC_RetVal_t Ret;
K12_REC_stEventFrame_t stFrameEvent;
// Fill Frame with data ....
...
long ltime;
time( &ltime );  // Gets current time from the PC
long lTSHigh, lTSLow;
Ret = K12_REC_ConvertGenericTimeToTimestamp(ltime, 0L, &lTSHigh,
&lTSLow);
if(Ret == K12_REC_ERROR)
   return;

stFrameEvent.ulTimestampHigh = (unsigned long) lTSHigh;
stFrameEvent.ulTimestampLow = (unsigned long) lTSLow;
Ret = K12_REC_Write(iWriteHandle, (K12_REC_stEventHead_t *)
&stFrameEvent);
```

See also:
K12_REC_ConvertTimestampToGenericTime

Library Information Commands

int K12_REC_GetDLLVersion ()

Description Returns the version number of the DLL.

Parameters none

Return Value The version of this DLL (for example 100 means V1.00)

Function Prototyp K12_REC_PROC_GET_DLL_VERSION

Appendix A: Abbreviations

AP
Application Processor

CEPT
Conférence Européenne des Administrations des Postes et des Télécommunications

CRC
Cyclic Redundance Check

DLL
Dynamic Link Library

L1
Layer One (Physical Layer)

LDS
Logical Data Source

LL
Logical Link

LSBF
Least Significant Byte First

MSBF
Most Significant Byte First

PLL
Phase Lock Loop

TRAU
Transcoder / Rate Adapter Unit

Index of Commands