

TCN*Open*

Open Source Development



TRDP

Train Real Time Data Protocol

TRDP SPY - Reference Manual

Contents

1	TRDP-SPY	1
1.1	Introduction	1
1.1.1	Purpose	1
1.1.2	Intended Audience	1
1.2	Design Description	1
1.2.1	System	1
1.2.2	Operational Environment	1
1.3	Development Environment for Windows	1
1.3.1	Steps to compile for Windows	2
1.4	Development Environment for Linux	2
1.4.1	Steps to compile and install Wireshark on Linux:	2
1.5	Interface	3
1.6	Usecase	4
2	Module Index	5
2.1	Modules	5
3	Data Structure Index	7
3.1	Data Structures	7
4	File Index	9
4.1	File List	9
5	Module Documentation	11
5.1	Parsing	11
5.1.1	Function Documentation	11
5.1.1.1	trdp_lookupType	11
5.1.1.2	trdp_parsebody_clean	12
5.1.1.3	trdp_parsebody_init	12
5.1.1.4	trdp_parsebody_isinited	12
5.1.1.5	trdp_parsebody_lookup	12

5.1.1.6	trdp_parsebody_search	12
5.2	Wireshark	14
5.2.1	Function Documentation	14
5.2.1.1	dissect_trdp	14
5.2.1.2	dissect_trdp_generic_body	14
5.2.1.3	proto_reg_handoff_trdp	15
5.2.1.4	proto_register_trdp	15
5.3	Definitions	16
5.3.1	Function Documentation	17
5.3.1.1	trdp_dissect_width	17
5.3.1.2	trdp_fcs32	17
6	Data Structure Documentation	19
6.1	ComId Struct Reference	19
6.1.1	Detailed Description	19
6.2	Dataset Struct Reference	20
6.2.1	Detailed Description	20
6.2.2	Field Documentation	20
6.2.2.1	listOfElements	20
6.3	Element Struct Reference	20
6.3.1	Detailed Description	21
6.3.2	Field Documentation	21
6.3.2.1	array_size	21
6.3.2.2	offset	22
6.3.2.3	type	22
7	File Documentation	23
7.1	lookuptype.h File Reference	23
7.1.1	Detailed Description	23
7.2	packet-trdp_spy.h File Reference	24
7.2.1	Detailed Description	24
7.3	parsebody.h File Reference	25
7.3.1	Detailed Description	26
7.4	trdp_env.h File Reference	26
7.4.1	Detailed Description	28

Chapter 1

TRDP-SPY

1.1 Introduction

1.1.1 Purpose

As part of the IP-Train project, two new protocols namely TRDP-PD (Process Data) and TRDP-MD (Message Data) are intended to be supported by the Wireshark tool. The support is envisaged to be made available in the form of a plug-in.

The existing GUI of the Wireshark V1.8.3 shall not be modified. The plug-in TRDP-SPY shall be available as a DLL for Windows platform and shared library for TRDP-spy for Linux platform.

1.1.2 Intended Audience

The TRDP-SPY will be used primarily by TRDP Engineers.

1.2 Design Description

1.2.1 System

TRDP Wire Protocol Analysis tool (TRDP-SPY) shall provide qualitative and quantitative analysis of TRDP streams, in order to verify system behaviour during qualification tests (level 2 and level 3) and provide help in problem analysis during train integration and debugging.

1.2.2 Operational Environment

The plug-in shall be compatible with Windows XP and Linux implementation of Wireshark. Standard behavior of Wireshark for all other protocols than WP shall not be influenced in any way by the TRDPWP analysis plug-in.

The plug-in shall be delivered as a DLL (Windows) along with the Wireshark-setup.exe and shared Library (.la, .lai and .so files or Linux) along with the minimal source - Wireshark-1.8.3.

1.3 Development Environment for Windows

Following specifications are used for development of the TRDP PD and TRDP MD plug-in for Wireshark.

- Operating System: Windows XP
- Tool : Wireshark V1.8.3
- Programming Language: C
- TRDP Wire Protocol

1.3.1 Steps to compile for Windows

Prerequisites:

- Wireshark minimal source (wireshark-1.8.3.tar.bz2).
- TRDP-SPY_src.zip source.
- Follow the online guide http://www.wireshark.org/docs/wsdg_html_chunked/ChSetupWin32.html

Steps:

- Unzip `wireshark-1.8.3.tar.bz2` to `c:\` and rename it to `wireshark`.
- Unzip `TRDP-SPY_src.zip`.
- From `TRDP-SPY_src` source copy folders to `c:/wireshark/plugins`.
- Also copy `config.nmake` from `TRDP-SPY` to `c:\wireshark` (overwrite the existing one).
- Open a Terminal and navigate to the plugin location `C:\wireshark\plugins\trpd_spy\`.
- First clean it using command `nmake -f makefile.nmake distclean` or run `clean.bat`.
- Then compile using command `nmake -f makefile.nmake` or run `build.bat`.

This will generate the `TRDP_spy.dll`.

The Wireshark version containing TRDP can be started with:

`C:\wireshark\wireshark-gtk2\wireshark.exe`

1.4 Development Environment for Linux

Following specifications are used for development of the TRDP PD and TRDP MD plug-in for Wireshark.

- Operating System: Ubuntu 12.04 LTS-Linux
- Tool : Wireshark v 1.8.3
- Programming Language: C

1.4.1 Steps to compile and install Wireshark on Linux:

Prerequisites:

- Wireshark source (`wireshark-1.8.3.tar.bz`).
- TRDP-SPY_src.zip.

Steps: Unzip wireshark-1.8.3.tar.bz with command on Console and not by using tar to unzip:

```
$ tar xjvf Wireshark-1.8.3.tar.bz
```

Now execute the following commands to compile and install Wireshark

```
$ cd wireshark-1.8.3
$ ./configure wireshark-prefix=/opt/local
$ make
$ make install
$ /opt/local/bin/wireshark (to launch Wireshark)
```

Integrate TRDP-SPY into this wireshark version.

Unzip TRDP-SPY.zip. Now copy folder TRDP_spy location wireshark-1.8.3/plugins/ and compile the plugin with the give following commands.

```
$ cd ../wireshark-1.8.3/plugins/trdp_spy
$ make clean
$ make
```

Please refer wireshark-1.8.3/readme and wireshark-1.8.3/install for more reference.

1.5 Interface

The plug-in shall be delivered as a DLL i.e. TRDP_spy.dll for Windows platform and shared library packet-trdp_spy.so files for Linux platform. For Application Data decoding additional libxml2.dll for Windows and libxml2.a, libxml2.la and libxml2.so for Linux are required which functions to parse the TRDP_config.xml file that contains the details of the Data-sets corresponding to each frame that is captured or logged by Wireshark.

Overall interface of the system can be explained as shown in the figure below:

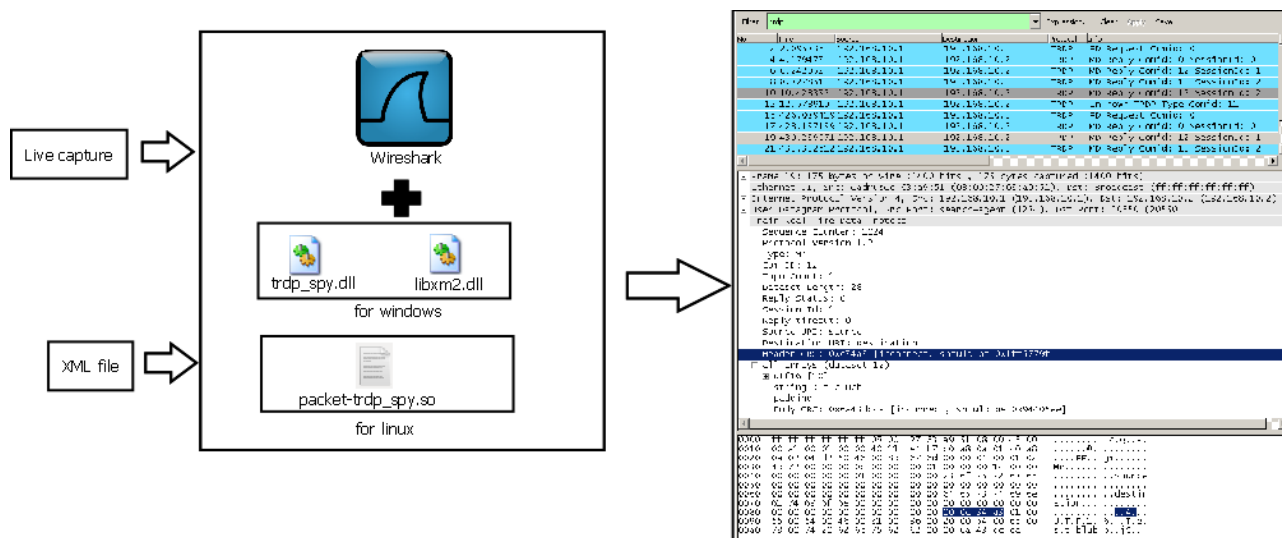


Figure 1.1: Interface Diagram

1.6 Usecase

The TRDP-SPY plugin is interated into Wireshark as described:

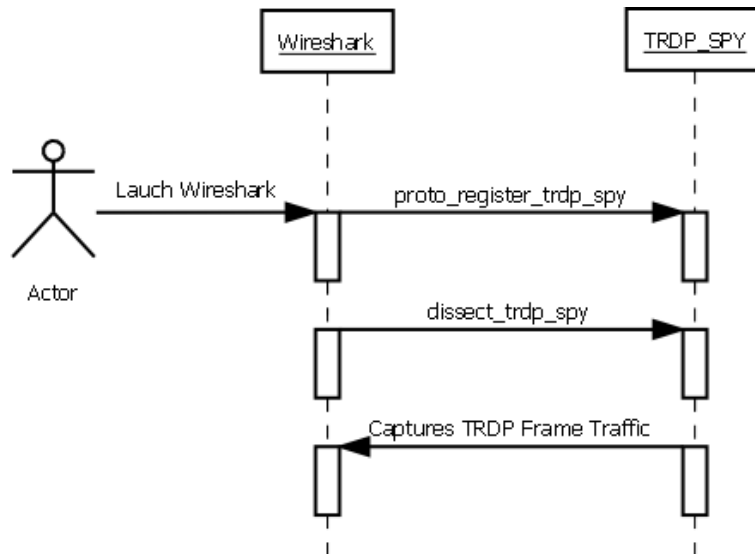


Figure 1.2: Live Functionality Sequence Diagram

On startup the plugin is registered in Wireshark, so the corresponding TCP and UDP packets are sent to this plugin. Each fitting packet is analyzed by the `trdp_dissect`.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Parsing	11
Wireshark	14
Definitions	16

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

ComId	This struct makes a mapping between one comId and one dataset	19
Dataset	Description of one dataset	20
Element	Description of one element	20

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

lookuptype.h	Functionality to find the corresponding type id for a name	23
packet-trdp_spy.h	Interface between Wireshark and the TRDP analysis module	24
parsebody.h	Loading of the XML description	25
trdp_env.h	Definition of the TRDP constants and specific calculations	26

Chapter 5

Module Documentation

5.1 Parsing

Data Structures

- struct [ComId](#)
This struct makes a mapping between one comId and one dataset.
- struct [Dataset](#)
Description of one dataset.
- struct [Element](#)
description of one element

Functions

- gint [trdp_lookupType](#) (GHashTable *pTableDataset, GString *nameOfType)
Search in the given table at the names and return the found id.
- TRDP_RET_t [trdp_parsebody_init](#) (const char **xmlconfigFile)
Create the module and extract all the needed information from the configuration file.
- void [trdp_parsebody_clean](#) (void)
Clean used memory.
- int [trdp_parsebody_isinitied](#) (void)
Show the status, if the library is ready to use.
- struct [Dataset](#) * [trdp_parsebody_lookup](#) (guint32 comId)
Looks up the dataset for a given [ComId](#).
- struct [Dataset](#) * [trdp_parsebody_search](#) (guint32 datasetId)
Uses the second hashmap to find the struct [Dataset](#) for a given datasetid.

5.1.1 Function Documentation

5.1.1.1 gint trdp_lookupType (GHashTable * pTableDataset, GString * nameOfType)

Search in the given table at the names and return the found id.

Parameters

in	<i>pTableDataset</i>	table containing all types
in	<i>nameOfType</i>	textual description of a type, searching for

Returns

found identifier, or zero on errors

5.1.1.2 void trdp_parsebody_clean (void)

Clean used memory.

Release all the allocated memory, needed to store the given information.

Returns

nothing

5.1.1.3 TRDP_RET_t trdp_parsebody_init (const char ** xmlconfigFile)

Create the module and extract all the needed information from the configuration file.

Parameters

in	<i>xmlconfigFile</i>	path to the file containing the XML description of the TRDP packets.
----	----------------------	--

Returns

TRDP_PARSEBODY_OK when no errors occurred

5.1.1.4 int trdp_parsebody_isinitd (void)

Show the status, if the library is ready to use.

Returns

> 0 if the library is initialized, 0 if uninitialized

5.1.1.5 struct Dataset * trdp_parsebody_lookup (guint32 comId) [read]

Looks up the dataset for a given [ComId](#).

Parameters

in	<i>comId</i>	to search for.
----	--------------	----------------

Returns

NULL, when nothing was found.

5.1.1.6 struct Dataset * trdp_parsebody_search (guint32 datasetId) [read]

Uses the second hashmap to find the struct [Dataset](#) for a given datasetid.

Parameters

in	<i>datasetId</i>	the dataset we are searching for
----	------------------	----------------------------------

Returns

NULL, when nothing was found.

5.2 Wireshark

Functions

- void [proto_register_trdp](#) (void)
start analyzing TRDP packets
- void [proto_reg_handoff_trdp](#) (void)
Called, if the analysis of TRDP packets is stopped.
- guint32 [dissect_trdp_generic_body](#) (tvbuff_t *tvb, packet_info *pinfo, proto_tree *trdp_spy_tree, proto_tree *trdpRootNode, guint32 trdp_spy_comid, guint32 offset, guint length, guint8 flag_dataset, guint8 dataset_level)
Extract all information from the userdata (uses the parsebody module for unmarshalling)
- void [dissect_trdp](#) (tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree)
Code to analyze the actual TRDP packet.

5.2.1 Function Documentation

5.2.1.1 void dissect_trdp (tvb, packet_info * pinfo, proto_tree * tree)

Code to analyze the actual TRDP packet.

Parameters

<i>tvb</i>	buffer
<i>pinfo</i>	info for the packet
<i>tree</i>	to which the information are added

Returns

nothing

5.2.1.2 guint32 dissect_trdp_generic_body (tvb, packet_info * pinfo, proto_tree * trdp_spy_tree, proto_tree * trdpRootNode, guint32 trdp_spy_comid, guint32 offset, guint length, guint8 flag_dataset, guint8 dataset_level)

Extract all information from the userdata (uses the parsebody module for unmarshalling)

Parameters

<i>tvb</i>	buffer
<i>packet</i>	info for the packet
<i>tree</i>	to which the information are added
<i>trdpRootNode</i>	Root node of the view of an TRDP packet (Necessary, as this function will be called recursively)
<i>trdp_spy_comid</i>	the already extracted comId
<i>offset</i>	where the userdata starts in the TRDP package
<i>flag_dataset</i>	on 0, the comId will be searched, on > 0 trdp_spy_comid will be interpreted as a dataset id
<i>dataset_level</i>	is set to 0 for the beginning

Returns

the actual offset in the package

5.2.1.3 void proto_reg_handoff_trdp (void)

Called, if the analysis of TRDP packets is stopped.

If this dissector uses sub-dissector registration add a registration routine. This exact format is required because a script is used to find these routines and create the code that calls these routines.

This function is also called by preferences whenever "Apply" is pressed (see prefs_register_protocol above) so it should accommodate being called more than once.

5.2.1.4 void proto_register_trdp (void)

start analyzing TRDP packets

Register the protocol with Wireshark this format is required because a script is used to build the C function that calls all the protocol registration.

5.3 Definitions

Defines

- #define **TRDP_BOOL8** 1
=UINT8, 1 bit relevant (equal to zero -> false, not equal to zero -> true)
- #define **TRDP_CHAR8** 2
char, can be used also as UTF8
- #define **TRDP_UTF16** 3
Unicode UTF-16 character.
- #define **TRDP_INT8** 4
Signed integer, 8 bit.
- #define **TRDP_INT16** 5
Signed integer, 16 bit.
- #define **TRDP_INT32** 6
Signed integer, 32 bit.
- #define **TRDP_INT64** 7
Signed integer, 64 bit.
- #define **TRDP_UINT8** 8
Unsigned integer, 8 bit.
- #define **TRDP_UINT16** 9
Unsigned integer, 16 bit.
- #define **TRDP_UINT32** 10
Unsigned integer, 32 bit.
- #define **TRDP_UINT64** 11
Unsigned integer, 64 bit.
- #define **TRDP_REAL32** 12
Floating point real, 32 bit.
- #define **TRDP_REAL64** 13
Floating point real, 64 bit.
- #define **TRDP_TIMEDATE32** 14
32 bit UNIX time
- #define **TRDP_TIMEDATE48** 15
48 bit TCN time (32 bit seconds and 16 bit ticks)
- #define **TRDP_TIMEDATE64** 16
32 bit seconds and 32 bit microseconds
- #define **TRDP_MD_HEADERLENGTH** TRDP_HEADER_MD_OFFSET_DATA
Length of the TRDP header of an MD message.
- #define **TRDP_FCS_LENGTH** 4
The CRC calculation results in a 32bit result so 4 bytes are necessary.

Functions

- quint32 **trdp_fcs32** (const quint8 buf[], quint32 len, quint32 fcs)
Compute crc32 according to IEEE802.3.
- quint8 **trdp_dissect_width** (quint32 type)
Lookup table for length of the standard types.

5.3.1 Function Documentation

5.3.1.1 guint8 trdp_dissect_width (guint32 type)

Lookup table for length of the standard types.

The width of an element in bytes. Extracted from table3 at TCN-TRDP2-D-BOM-011-19.

Parameters

in	type	the numeric representation of a type
----	------	--------------------------------------

Returns

the width in byte of one element of the given type

5.3.1.2 guint32 trdp_fcs32 (const guint8 buf[], guint32 len, guint32 fcs)

Compute crc32 according to IEEE802.3.

Note

Returned CRC is inverted

Parameters

in	buf	Input buffer
in	len	Length of input buffer
in	fcs	Initial (seed) value for the FCS calculation

Returns

Calculated fcs value

Chapter 6

Data Structure Documentation

6.1 ComId Struct Reference

This struct makes a mapping between one comId and one dataset.

```
#include <parsebody.h>
```

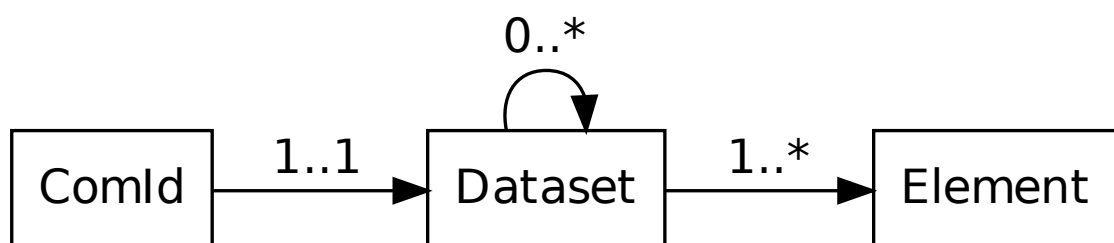
Data Fields

- guint32 [comId](#)
Communication Id, used as key.
- guint [datasetId](#)
Id for a dataset ([see Dataset structure](#))

6.1.1 Detailed Description

This struct makes a mapping between one comId and one dataset.

The following overview visualizes the relation between one comId and an element of a dataset:



There is a separate structure for datasets necessary, because the dataset itself can be packed recursively into each other.

The documentation for this struct was generated from the following file:

- [parsebody.h](#)

6.2 Dataset Struct Reference

Description of one dataset.

```
#include <parsebody.h>
```

Data Fields

- quint [datasetId](#)
Unique identification of one dataset.
- GString * [name](#)
Description of the dataset.
- GSList * [listOfElements](#)
All elements, this dataset consists of.

6.2.1 Detailed Description

Description of one dataset.

6.2.2 Field Documentation

6.2.2.1 GSList* Dataset::listOfElements

All elements, this dataset consists of.

The documentation for this struct was generated from the following file:

- [parsebody.h](#)

6.3 Element Struct Reference

description of one element

```
#include <parsebody.h>
```

Data Fields

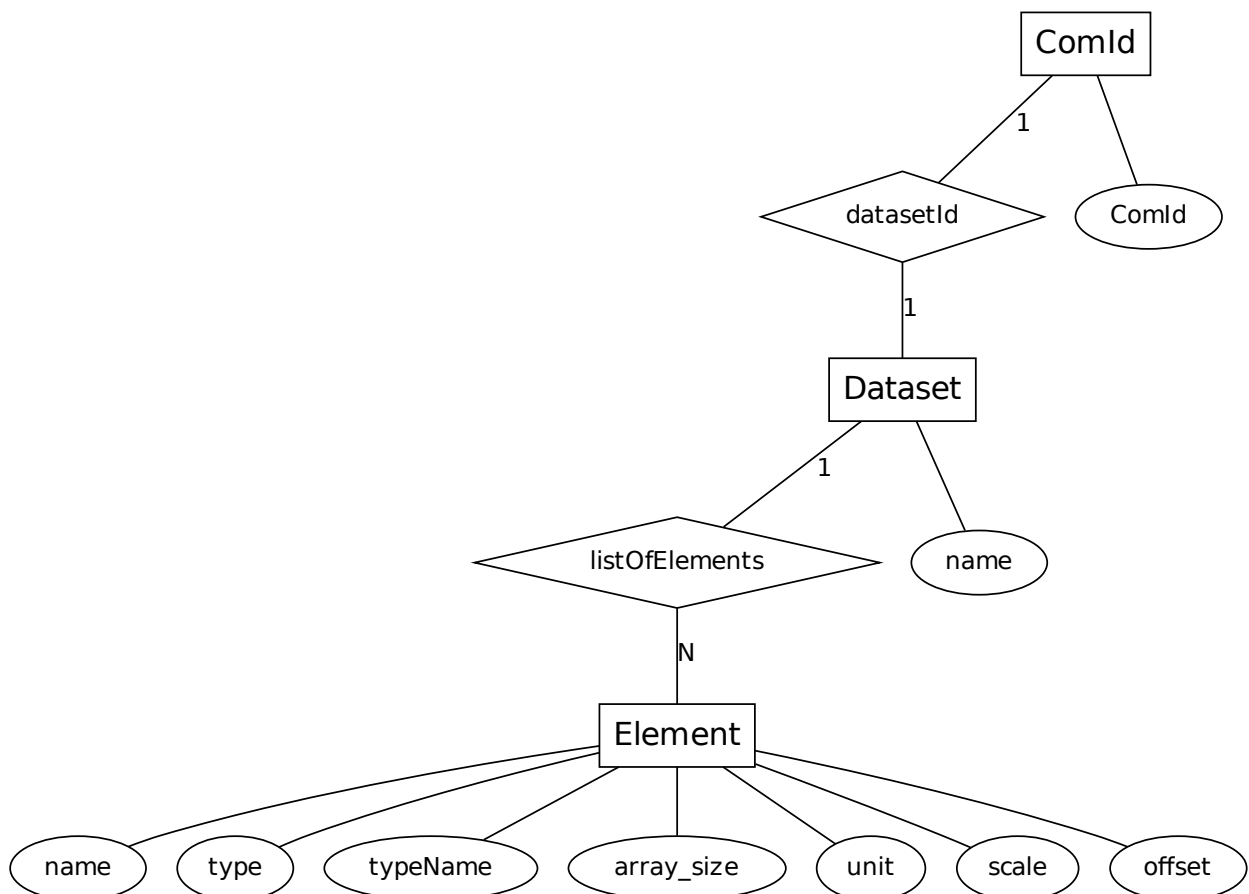
- GString * [name](#)
Name of the variable, that is stored.
- quint32 [type](#)
Numeric type of the variable (see Usermanual, chapter 4.2) or defined at [TRDP_BOOL8](#), [TRDP_UINT8](#), [TRDP_UINT16](#) and so on.
- GString * [typeName](#)
Textual representation of the type (necessary for own datasets, packed recursively)
- quint32 [array_size](#)

- Amount this value occurred.
- GString * [unit](#)
Unit to display.
- gfloat [scale](#)
A factor the given value is scaled.
- gint32 [offset](#)
Offset that is added to the values.

6.3.1 Detailed Description

description of one element

All persisted information can be seen in this diagram:



6.3.2 Field Documentation

6.3.2.1 guint32 Element::array_size

Amount this value occurred.

1 is default; 0 indicates a dynamic list (the dynamic list starts with a 16bit value with the occurrence)

6.3.2.2 gint32 Element::offset

Offset that is added to the values.

displayed value = scale * raw value + offset

6.3.2.3 guint32 Element::type

Numeric type of the variable (see Usermanual, chapter 4.2) or defined at [TRDP_BOOL8](#), [TRDP_UINT8](#), [TRDP_UINT16](#) and so on.

The documentation for this struct was generated from the following file:

- [parsebody.h](#)

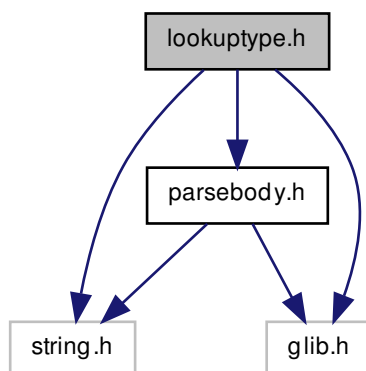
Chapter 7

File Documentation

7.1 lookuptype.h File Reference

Functionality to find the corresponding type id for a name.

```
#include <string.h> #include <glib.h> #include "parsebody.h" Include dependency graph for lookuptype.h:
```



Functions

- gint [trdp_lookupType](#) (GHashTable *pTableDataset, GString *nameOfType)

Search in the given table at the names and return the found id.

7.1.1 Detailed Description

Functionality to find the corresponding type id for a name.

Note

Project: TRDP SPY

Author

Florian Weispfenning, Bombardier Transportation

Remarks

This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at <http://mozilla.org/MPL/2.0/>. Copyright Bombardier Transportation Inc. or its subsidiaries and others, 2013. All rights reserved.

Id:

[lookuptype.h](#) 1154 2014-01-21 13:11:58Z fweispf

7.2 packet-trdp_spy.h File Reference

Interface between Wireshark and the TRDP analysis module.

Functions

- void [proto_register_trdp](#) (void)
start analyzing TRDP packets
- void [proto_reg_handoff_trdp](#) (void)
Called, if the analysis of TRDP packets is stopped.
- guint32 [dissect_trdp_generic_body](#) (tvbuff_t *tvb, packet_info *pinfo, proto_tree *trdp_spy_tree, proto_tree *trdpRootNode, guint32 trdp_spy_comid, guint32 offset, guint length, guint8 flag_dataset, guint8 dataset_level)
Extract all information from the userdata (uses the parsebody module for unmarshalling)
- void [dissect_trdp](#) (tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree)
Code to analyze the actual TRDP packet.

7.2.1 Detailed Description

Interface between Wireshark and the TRDP analysis module.

Note

Project: TRDP SPY

Author

Florian Weispfenning, Bombardier Transportation

Remarks

This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at <http://mozilla.org/MPL/2.0/>. Copyright Bombardier Transportation Inc. or its subsidiaries and others, 2013. All rights reserved.

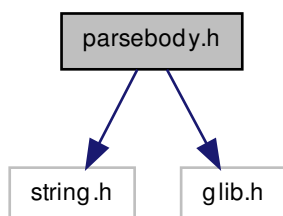
Id:

[packet-trdp_spy.h](#) 1167 2014-02-17 15:02:13Z fweispf

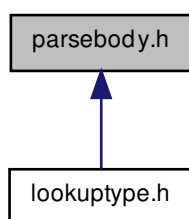
7.3 parsebody.h File Reference

Loading of the XML description.

#include <string.h> #include <glib.h> Include dependency graph for parsebody.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [ComId](#)
This struct makes a mapping between one comId and one dataset.
- struct [Dataset](#)
Description of one dataset.
- struct [Element](#)
description of one element

Functions

- TRDP_RET_t [trdp_parsebody_init](#) (const char **xmlconfigFile)
Create the module and extract all the needed information from the configuration file.
- void [trdp_parsebody_clean](#) (void)

Clean used memory.

- int `trdp_parsebody_isinited` (void)

Show the status, if the library is ready to use.

- struct `Dataset` * `trdp_parsebody_lookup` (quint32 comId)

Looks up the dataset for a given ComId.

- struct `Dataset` * `trdp_parsebody_search` (quint32 datasetId)

Uses the second hashmap to find the struct `Dataset` for a given datasetId.

7.3.1 Detailed Description

Loading of the XML description.

Note

Project: TRDP SPY

Author

Florian Weispfenning, Bombardier Transportation

Remarks

This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at <http://mozilla.org/MPL/2.0/>. Copyright Bombardier Transportation Inc. or its subsidiaries and others, 2013. All rights reserved.

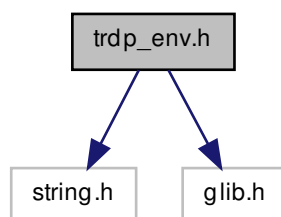
Id:

`parsebody.h` 1176 2014-02-27 15:47:25Z fweispf

7.4 trdp_env.h File Reference

Definition of the TRDP constants and specific calculations.

`#include <string.h> #include <glib.h>` Include dependency graph for `trdp_env.h`:



Defines

- #define [TRDP_BOOL8](#) 1
=UINT8, 1 bit relevant (equal to zero -> false, not equal to zero -> true)
- #define [TRDP_CHAR8](#) 2
char, can be used also as UTF8
- #define [TRDP_UTF16](#) 3
Unicode UTF-16 character.
- #define [TRDP_INT8](#) 4
Signed integer, 8 bit.
- #define [TRDP_INT16](#) 5
Signed integer, 16 bit.
- #define [TRDP_INT32](#) 6
Signed integer, 32 bit.
- #define [TRDP_INT64](#) 7
Signed integer, 64 bit.
- #define [TRDP_UINT8](#) 8
Unsigned integer, 8 bit.
- #define [TRDP_UINT16](#) 9
Unsigned integer, 16 bit.
- #define [TRDP_UINT32](#) 10
Unsigned integer, 32 bit.
- #define [TRDP_UINT64](#) 11
Unsigned integer, 64 bit.
- #define [TRDP_REAL32](#) 12
Floating point real, 32 bit.
- #define [TRDP_REAL64](#) 13
Floating point real, 64 bit.
- #define [TRDP_TIMEDATE32](#) 14
32 bit UNIX time
- #define [TRDP_TIMEDATE48](#) 15
48 bit TCN time (32 bit seconds and 16 bit ticks)
- #define [TRDP_TIMEDATE64](#) 16
32 bit seconds and 32 bit microseconds
- #define [TRDP_MD_HEADERLENGTH](#) [TRDP_HEADER_MD_OFFSET_DATA](#)
Length of the TRDP header of an MD message.
- #define [TRDP_FCS_LENGTH](#) 4
The CRC calculation results in a 32bit result so 4 bytes are necessary.

Functions

- guint32 [trdp_fcs32](#) (const guint8 buf[], guint32 len, guint32 fcs)
Compute crc32 according to IEEE802.3.
- guint8 [trdp_dissect_width](#) (guint32 type)
Lookup table for length of the standard types.

7.4.1 Detailed Description

Definition of the TRDP constants and specific calculations.

Note

Project: TRDP SPY

Author

Florian Weispfenning, Bombardier Transportation

Remarks

This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at <http://mozilla.org/MPL/2.0/>. Copyright Bombardier Transportation Inc. or its subsidiaries and others, 2013. All rights reserved.

Id: