## Btparser

A program failure analysis library

Karel Klíč

November 2, 2012

# **Contents**

1	Ove	rview	9
Ι	Coi	ncepts	11
2	Stac	k Trace Normalization	13
3	Stac	k Trace Clustering	15
4	Core	e Dump Failure Analysis	17
5	Wisl	hlist	19
II	Im	nplementation	21
6	Ove	rview	23
7	Data	a Structure Index	25
	7.1	Data Structures	25
	7.2	File List	26
8	Data	a Structure Documentation	27
	8.1	btp_callgraph Struct Reference	27
		8.1.1 Detailed Description	27
		8.1.2 Field Documentation	27
	8.2	btp_cluster Struct Reference	29
		8.2.1 Detailed Description	29
	8.3	btp_core_frame Struct Reference	30
		8.3.1 Detailed Description	30
		8.3.2 Field Documentation	30
	0.4	hts and stalling Charles Defende	20

	8.4.1	Detailed Description	32
	8.4.2	Field Documentation	32
8.5	btp_co	re_thread Struct Reference	34
	8.5.1	Detailed Description	34
	8.5.2	Field Documentation	34
8.6	btp_de	b_package Struct Reference	35
8.7	btp_de	ndrogram Struct Reference	36
	8.7.1	Detailed Description	36
	8.7.2	Field Documentation	36
8.8	btp_dis	stances Struct Reference	37
	8.8.1	Detailed Description	37
8.9	btp_elf	_fde Struct Reference	38
	8.9.1	Detailed Description	38
	8.9.2	Field Documentation	38
8.10	btp_elf	_plt_entry Struct Reference	39
	8.10.1	Detailed Description	39
	8.10.2	Field Documentation	39
8.11	btp_gd	b_frame Struct Reference	40
	8.11.1	Detailed Description	40
	8.11.2	Field Documentation	40
8.12	btp_gd	b_sharedlib Struct Reference	42
	8.12.1	Detailed Description	42
8.13	btp_gd	b_stacktrace Struct Reference	43
	8.13.1	Detailed Description	43
	8.13.2	Field Documentation	43
8.14	btp_gd	b_thread Struct Reference	44
	8.14.1	Detailed Description	44
	8.14.2	Field Documentation	44
8.15	btp_jsc	on_settings Struct Reference	45
8.16	btp_jsc	on_value Struct Reference	46
8.17	btp_ko	ops_frame Struct Reference	47
	8.17.1	Detailed Description	47
	8.17.2	Field Documentation	47
8.18	btp_ko	ops_stacktrace Struct Reference	49
	8.18.1	Field Documentation	49
8.19	btp loc	eation Struct Reference	51

		8.19.1 Detailed Description	51
		8.19.2 Field Documentation	51
	8.20	btp_operating_system Struct Reference	52
	8.21	1 btp_python_frame Struct Reference	53
	8.22	2 btp_python_stacktrace Struct Reference	54
	8.23	3 btp_report Struct Reference	55
	8.24	4 btp_rpm_consistency Struct Reference	56
	8.25	5 btp_rpm_package Struct Reference	57
	8.26	5 btp_sha1_state Struct Reference	58
		8.26.1 Detailed Description	58
	8.27	7 btp_strbuf Struct Reference	59
		8.27.1 Detailed Description	59
		8.27.2 Field Documentation	59
	8.28	B btp_unstrip_entry Struct Reference	60
		8.28.1 Detailed Description	60
9	File l	e Documentation	61
	9.1	callgraph.h File Reference	
	<i>,</i> ,,,	9.1.1 Detailed Description	
		9.1.2 Function Documentation	
	9.2	cluster.h File Reference	
		9.2.1 Detailed Description	
		9.2.2 Function Documentation	
	9.3	core_fingerprint.h File Reference	
		9.3.1 Detailed Description	
	9.4	core_frame.h File Reference	
		9.4.1 Detailed Description	
		9.4.2 Function Documentation	
	9.5	core_stacktrace.h File Reference	
		9.5.1 Detailed Description	
		9.5.2 Function Documentation	
	9.6	core_thread.h File Reference	71
		9.6.1 Detailed Description	71
		9.6.2 Function Documentation	
	9.7	deb.h File Reference	73
		9.7.1 Detailed Description	73
	9.8	disasm.h File Reference	74

	9.8.1	Detailed Description	74
	9.8.2	Function Documentation	74
9.9	elves.h	n File Reference	75
	9.9.1	Detailed Description	75
	9.9.2	Function Documentation	75
9.10	gdb_fra	rame.h File Reference	77
	9.10.1	Detailed Description	78
	9.10.2	Function Documentation	78
9.11	gdb_sh	haredlib.h File Reference	84
	9.11.1	Detailed Description	84
	9.11.2	Function Documentation	84
9.12	gdb_sta	tacktrace.h File Reference	87
	9.12.1	Detailed Description	87
	9.12.2	Function Documentation	88
9.13	gdb_th	nread.h File Reference	93
	9.13.1	Detailed Description	93
	9.13.2	Function Documentation	94
9.14	koops_	_frame.h File Reference	98
	9.14.1	Detailed Description	98
	9.14.2	Function Documentation	98
9.15	koops_	_stacktrace.h File Reference	101
	9.15.1	Detailed Description	101
	9.15.2	Function Documentation	101
9.16	location	on.h File Reference	103
	9.16.1	Detailed Description	103
	9.16.2	Function Documentation	103
9.17	metrics	s.h File Reference	106
	9.17.1	Detailed Description	107
	9.17.2	Typedef Documentation	107
	9.17.3	Function Documentation	107
9.18	normal	lize.h File Reference	109
	9.18.1	Detailed Description	109
	9.18.2	Function Documentation	109
9.19	python	n_frame.h File Reference	110
	9.19.1	Detailed Description	110
	9.19.2	Function Documentation	110

	0.20	python_stacktrace.h File Reference	112
	9.20		
		9.20.1 Detailed Description	112
		9.20.2 Function Documentation	112
	9.21	rpm.h File Reference	114
		9.21.1 Detailed Description	114
		9.21.2 Function Documentation	114
	9.22	sha1.h File Reference	115
		9.22.1 Detailed Description	115
	9.23	strbuf.h File Reference	116
		9.23.1 Detailed Description	116
		9.23.2 Function Documentation	116
	9.24	unstrip.h File Reference	119
		9.24.1 Detailed Description	119
	9.25	utils.h File Reference	120
		9.25.1 Detailed Description	121
		9.25.2 Function Documentation	121
		9.25.3 Variable Documentation	125
4.0	_		
10	Exan	nple Documentation	127
	10.1	/home/karel/devel/btparser/lib/koops_frame.h	127
11	Knov	wn Bugs	129
11	12110	mir Dugo	14)
12	Wish	nlist	131
	_		
Ind	lex		132

## **Overview**

Failures of computer programs are omnipresent in the information technology industry: they occur during software development, software testing, and also in production. Failures occur in programs from all levels of the system stack. The program environment differ substantially between kernel space, user space programs written in C or C++, Python scripts, and Java applications, but the general structure of failures is surprisingly similar between the mentioned environments due to imperative nature of the languages and common concepts such as procedures, objects, exceptions.

Btparser is a collection of low-level algorithms for program failure processing, analysis, and reporting supporting kernel space, user space, Python, and Java programs. Considering failure processing, it allows to parse failure description from various sources such as GDB-created stack traces, Python stack traces with a description of uncaught exception, and kernel oops message. Infromation can also be extracted from the core dumps of unexpectedly terminated user space processes and from the machine executable code of binaries. Considering failure analysis, the stack traces of failed processes can be normalized, trimmed, and compared. Clusters of similar stack traces can be calculated. In multi-threaded stack traces, the threads that caused the failure can be discovered. Considering failure reporting, the library can generate a failure report in a well-specified format, and the report can be sent to a remote machine.

Due to the low-level nature of the library and implementors' use cases, most of its functionality is currently limited to Linux-based operating systems using ELF binaries. The library can be extended to support Microsoft Windows and OS X platforms without changing its design, but dedicated engineering effort would be required to accomplish that.

10 Overview

# Part I Concepts

# **Stack Trace Normalization**

# **Stack Trace Clustering**

# **Core Dump Failure Analysis**

# Wishlist

Security Impact.

ABI compatibility check.

Collecting environment data.

<u>20</u> Wishlist

# Part II Implementation

# **Overview**

Btparser is implemented in the C language as defined in the C99 standard (ISO/IEC 9899:1999). It uses the C standard library and some additional libraries. No additional library is mandatory, though. When a library is not found by the build configuration script, the features requiring that library become unavailable. This approach improves both usability and portability of the library.

24 Overview

# **Data Structure Index**

### 7.1 Data Structures

Here are the data structures with brief descriptions:

btp_callgraph (A call graph representing calling relationships between subroutines )	27
btp_cluster (A cluster of objects from a dendrogram )	29
btp_core_frame (A function call on call stack of a core dump )	30
btp_core_stacktrace (A stack trace of a core dump )	32
btp_core_thread (A thread of execution on call stack of a core dump )	34
btp_deb_package	35
btp_dendrogram (A dendrogram created by clustering )	36
btp_distances (A distance matrix of stack trace threads )	37
btp_elf_fde (A single Frame Description Entry of the .eh_frame section present in ELF binaries )	38
btp_elf_plt_entry (A single item of the Procedure Linkage Table present in ELF binaries )	39
btp_gdb_frame (A function call of a GDB-produced stack trace )	40
btp_gdb_sharedlib (A shared library memory location as reported by GDB )	42
btp_gdb_stacktrace (A stack trace produced by GDB )	43
btp_gdb_thread (A thread of execution of a GDB-produced stack trace )	44
btp_json_settings	45
btp_json_value	46
btp_koops_frame (Kernel oops stack frame )	47
btp_koops_stacktrace	49
btp_location (A location of a parser in the input stream )	51
btp_operating_system	52
btp_python_frame	53
btp_python_stacktrace	54
btp_report	55
btp_rpm_consistency	56
btp_rpm_package	57
btp_sha1_state (Internal state of a SHA-1 hash algorithm )	58
btp_strbuf (A resizable string buffer )	59
btp_unstrip_entry (Core dump memory layout as reported by the unstrip utility )	60

26 Data Structure Index

## 7.2 File List

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

callgraph.h (Calling relationships between subroutines )
cluster.h (Clustering for stack trace threads )
config.h
core_fingerprint.h (Fingerprint algorithm for core stack traces )
core_frame.h (Single frame of core stack trace thread )
core_stacktrace.h (A stack trace of a core dump )
core_thread.h (Single thread of execution of a core stack trace )
core_unwind.h
deb.h (Deb-related structures and utilities )
disasm.h (BFD-based function disassembler )
elves.h (Loading PLT and FDEs from ELF binaries )
gdb_frame.h (Single frame of GDB stack trace thread )
gdb_sharedlib.h (Shared library information as produced by GDB )
gdb_stacktrace.h (Stack trace as produced by GDB )
gdb_thread.h (Single thread of execution of GDB stack trace )
json.h
koops_frame.h (Kernel oops stack frame )
koops_stacktrace.h (Kernel oops stack trace structure and related algorithms )
location.h (Parser location in input file )
metrics.h (Distance between stack trace threads )
normalize.h (Normalization of stack traces )
python_frame.h (Python frame structure and related algorithms )
python_stacktrace.h (Python stack trace structure and related algorithms )
report.h
rpm.h (RPM-related structures and utilities )
sha1.h (An implementation of SHA-1 cryptographic hash function )
strbuf.h (A string buffer structure and related algorithms )
unstrip.h (Parser for the output of the unstrip utility )
utils.h (Various utility functions, macros and variables that do not fit elsewhere)

## **Data Structure Documentation**

#### 8.1 btp\_callgraph Struct Reference

A call graph representing calling relationships between subroutines. #include <callgraph.h>Collaboration diagram for btp\_callgraph:



#### **Data Fields**

- uint64\_t address

  An offset to the start of a function executable code.
- uint64\_t \* callees

  A list of offsets to called functions.
- struct btp\_callgraph \* next

  Next node of the call graph or NULL.

#### **8.1.1 Detailed Description**

A call graph representing calling relationships between subroutines. It's a context-insensitive static call graph specialized to low-level programs. Functions are identified by their numeric address (an offset to a binary file).

#### 8.1.2 Field Documentation

#### 8.1.2.1 uint64\_t\* btp\_callgraph::callees

A list of offsets to called functions. It is terminated by a zero offset.

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

• callgraph.h

## 8.2 btp\_cluster Struct Reference

A cluster of objects from a dendrogram.

#include <cluster.h>Collaboration diagram for btp\_cluster:



#### **Data Fields**

- int size
- int \* objects
- struct btp\_cluster \* next

#### **8.2.1** Detailed Description

A cluster of objects from a dendrogram.

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

• cluster.h

#### 8.3 btp\_core\_frame Struct Reference

A function call on call stack of a core dump.

#include <core\_frame.h>Collaboration diagram for btp\_core\_frame:



#### **Data Fields**

- uint64\_t address
- char \* build\_id
- uint64 t build id offset
- char \* function name
- char \* file\_name
- char \* fingerprint
- struct btp\_core\_frame \* next

#### 8.3.1 Detailed Description

A function call on call stack of a core dump.

#### **8.3.2** Field Documentation

#### 8.3.2.1 uint64\_t btp\_core\_frame::address

Address of the machine code in memory. This is useful only when build\_id is not present for some reason. For example, this might be a null dereference (address is 0) or calling a method from null class pointer (address is a low number -- offset to the class).

Some programs generate machine code during runtime (JavaScript engines, JVM, the Gallium llvmpipe driver).

#### 8.3.2.2 char\* btp\_core\_frame::build\_id

Build id of the ELF binary. It might be NULL if the frame does not point to memory with code.

#### 8.3.2.3 char\* btp\_core\_frame::fingerprint

Hash of the function contents.

#### 8.3.2.4 struct btp\_core\_frame\* btp\_core\_frame::next [read]

A sibling frame residing below this one, or NULL if this is the last frame in the parent thread.

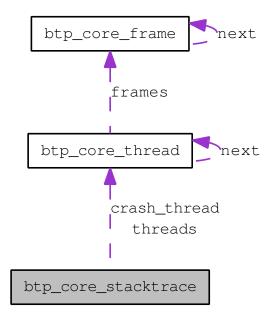
The documentation for this struct was generated from the following file:

• core\_frame.h

## 8.4 btp\_core\_stacktrace Struct Reference

A stack trace of a core dump.

#include <core\_stacktrace.h>Collaboration diagram for btp\_core\_stacktrace:



#### **Data Fields**

- uint8\_t signal
- char \* executable
- struct btp\_core\_thread \* crash\_thread Thread responsible for the crash.
- struct btp\_core\_thread \* **threads**

#### **8.4.1 Detailed Description**

A stack trace of a core dump.

#### **8.4.2** Field Documentation

#### 8.4.2.1 struct btp\_core\_thread\* btp\_core\_stacktrace::crash\_thread [read]

Thread responsible for the crash. It might be NULL if the crash thread is not detected.

#### 8.4.2.2 uint8\_t btp\_core\_stacktrace::signal

Signal number.

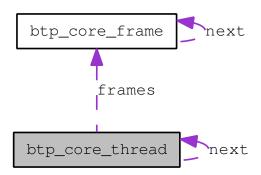
The documentation for this struct was generated from the following file:

• core\_stacktrace.h

## 8.5 btp\_core\_thread Struct Reference

A thread of execution on call stack of a core dump.

#include <core\_thread.h>Collaboration diagram for btp\_core\_thread:



#### **Data Fields**

- struct btp\_core\_frame \* frames
- struct btp\_core\_thread \* next

#### 8.5.1 Detailed Description

A thread of execution on call stack of a core dump.

#### 8.5.2 Field Documentation

#### 8.5.2.1 struct btp\_core\_frame\* btp\_core\_thread::frames [read]

Thread's frames, starting from the top of the stack.

#### 8.5.2.2 struct btp\_core\_thread\* btp\_core\_thread::next [read]

A sibling thread, or NULL if this is the last thread in a stacktrace.

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

• core\_thread.h

## 8.6 btp\_deb\_package Struct Reference

Collaboration diagram for btp\_deb\_package:



#### **Data Fields**

• struct btp\_deb\_package \* next

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

• deb.h

## 8.7 btp\_dendrogram Struct Reference

A dendrogram created by clustering.

#include <cluster.h>

#### **Data Fields**

- int size
- int \* order
- float \* merge\_levels

#### 8.7.1 Detailed Description

A dendrogram created by clustering.

#### 8.7.2 Field Documentation

#### $\textbf{8.7.2.1} \quad \textbf{float* btp\_dendrogram::merge\_levels}$

Levels at which the clusters were merged. The clustering can be reconstructed in order of increasing levels. There are (size - 1) levels.

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

• cluster.h

# 8.8 btp\_distances Struct Reference

A distance matrix of stack trace threads.

#include <metrics.h>

# **Data Fields**

- $\bullet$  int  $\mathbf{m}$
- int **n**
- float \* distances

# 8.8.1 Detailed Description

A distance matrix of stack trace threads. The distances are stored in a m-by-n two-dimensional array, where only entries (i, j) where i < j are actually stored.

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

• metrics.h

# 8.9 btp\_elf\_fde Struct Reference

A single Frame Description Entry of the .eh\_frame section present in ELF binaries.

#include <elves.h>Collaboration diagram for btp\_elf\_fde:



# **Data Fields**

- uint64\_t start\_address
- uint64\_t length
- struct btp\_elf\_fde \* next

# **8.9.1 Detailed Description**

A single Frame Description Entry of the .eh\_frame section present in ELF binaries.

### 8.9.2 Field Documentation

### 8.9.2.1 uint64\_t btp\_elf\_fde::length

Length of the function in bytes.

### 8.9.2.2 uint64\_t btp\_elf\_fde::start\_address

Offset where a function starts. If the function is present in the Procedure Linkage Table, this address matches some address in btp\_elf\_plt\_entry.

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

• elves.h

# 8.10 btp\_elf\_plt\_entry Struct Reference

A single item of the Procedure Linkage Table present in ELF binaries. #include <elves.h>Collaboration diagram for btp\_elf\_plt\_entry:



# **Data Fields**

- uint64\_t address
- char \* symbol\_name
- struct btp\_elf\_plt\_entry \* next

# 8.10.1 Detailed Description

A single item of the Procedure Linkage Table present in ELF binaries.

# 8.10.2 Field Documentation

# 8.10.2.1 uint64\_t btp\_elf\_plt\_entry::address

Address of the entry.

# $8.10.2.2 \quad char*\ btp\_elf\_plt\_entry::symbol\_name$

Symbol name corresponding to the address.

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

• elves.h

# 8.11 btp\_gdb\_frame Struct Reference

A function call of a GDB-produced stack trace.

#include <gdb\_frame.h>Collaboration diagram for btp\_gdb\_frame:



# **Data Fields**

- char \* function\_name
- char \* function\_type
- uint32\_t number
- char \* source\_file
- uint32\_t source\_line
- bool signal\_handler\_called
- uint64\_t address
- char \* library\_name
- struct btp\_gdb\_frame \* next

# 8.11.1 Detailed Description

A function call of a GDB-produced stack trace. A frame representing a function call or a signal handler on a call stack of a thread.

# 8.11.2 Field Documentation

### 8.11.2.1 uint64\_t btp\_gdb\_frame::address

The function address in the computer memory, or -1 when the address is unknown. Address is unknown when the frame represents inlined function.

# 8.11.2.2 char\* btp\_gdb\_frame::function\_name

A function name or NULL. If it's NULL, signal\_handler\_called is true.

# 8.11.2.3 char\* btp\_gdb\_frame::function\_type

A function type, or NULL if it isn't present.

### 8.11.2.4 char\* btp\_gdb\_frame::library\_name

A library name or NULL.

### 8.11.2.5 struct btp\_gdb\_frame\* btp\_gdb\_frame::next [read]

A sibling frame residing below this one, or NULL if this is the last frame in the parent thread.

### 8.11.2.6 uint32\_t btp\_gdb\_frame::number

A frame number in a thread. It does not necessarily show the actual position in the thread, as this number is set by the parser and never updated.

# 8.11.2.7 bool btp\_gdb\_frame::signal\_handler\_called

Signal handler was called on this frame.

# 8.11.2.8 char\* btp\_gdb\_frame::source\_file

The name of the source file containing the function definition, or the name of the binary file (.so) with the binary code of the function, or NULL.

### 8.11.2.9 uint32\_t btp\_gdb\_frame::source\_line

A line number in the source file, determining the position of the function definition, or -1 when unknown. The documentation for this struct was generated from the following file:

• gdb\_frame.h

# 8.12 btp\_gdb\_sharedlib Struct Reference

A shared library memory location as reported by GDB.

#include <gdb\_sharedlib.h>Collaboration diagram for btp\_gdb\_sharedlib:



# **Data Fields**

- uint64\_t from
- uint64\_t **to**
- int symbols
- char \* soname
- struct btp\_gdb\_sharedlib \* next

# **8.12.1** Detailed Description

A shared library memory location as reported by GDB.

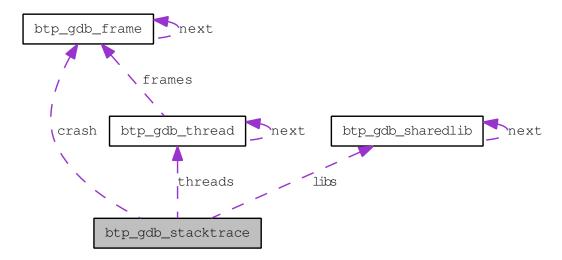
The documentation for this struct was generated from the following file:

• gdb\_sharedlib.h

# 8.13 btp\_gdb\_stacktrace Struct Reference

A stack trace produced by GDB.

#include <gdb\_stacktrace.h>Collaboration diagram for btp\_gdb\_stacktrace:



### **Data Fields**

- struct btp\_gdb\_thread \* threads
- struct btp\_gdb\_frame \* crash
- struct btp\_gdb\_sharedlib \* libs

# **8.13.1** Detailed Description

A stack trace produced by GDB. A stacktrace obtained at the time of a program crash, consisting of several threads which contains frames.

This structure represents a stacktrace as produced by the GNU Debugger.

## 8.13.2 Field Documentation

# 8.13.2.1 struct btp\_gdb\_frame\* btp\_gdb\_stacktrace::crash [read]

The frame where the crash happened according to debugger. It might be that we can not tell to which thread this frame belongs, because some threads end with mutually indistinguishable frames.

### 8.13.2.2 struct btp\_gdb\_sharedlib\* btp\_gdb\_stacktrace::libs [read]

Shared libraries loaded at the moment of crash.

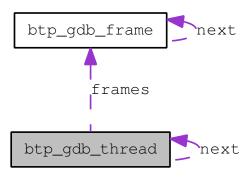
The documentation for this struct was generated from the following file:

• gdb\_stacktrace.h

# 8.14 btp\_gdb\_thread Struct Reference

A thread of execution of a GDB-produced stack trace.

#include <gdb\_thread.h>Collaboration diagram for btp\_gdb\_thread:



### **Data Fields**

- uint32\_t number
- struct btp\_gdb\_frame \* frames
- struct btp\_gdb\_thread \* next

# 8.14.1 Detailed Description

A thread of execution of a GDB-produced stack trace. Represents a thread containing frames.

### 8.14.2 Field Documentation

# 8.14.2.1 struct btp\_gdb\_frame\* btp\_gdb\_thread::frames [read]

Thread's frames, starting from the top of the stack.

# 8.14.2.2 struct btp\_gdb\_thread\* btp\_gdb\_thread::next [read]

A sibling thread, or NULL if this is the last thread in a stacktrace.

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

• gdb\_thread.h

# 8.15 btp\_json\_settings Struct Reference

# **Data Fields**

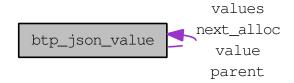
- unsigned long max\_memory
- int settings

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

• json.h

# 8.16 btp\_json\_value Struct Reference

Collaboration diagram for btp\_json\_value:



# **Data Fields**

```
• struct btp_json_value * parent
• enum btp_json_type type
• union {
    int boolean
    long integer
    double dbl
    struct {
       unsigned length
       char * ptr
    } string
    struct {
       unsigned length
       struct {
         char * name
         struct btp_json_value * value
       } values
    } object
    struct {
       unsigned length
       struct btp_json_value ** values
    } array
  } u
• union {
    struct btp_json_value * next_alloc
    void * object_mem
  } _reserved
```

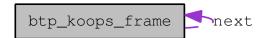
The documentation for this struct was generated from the following file:

• json.h

# 8.17 btp\_koops\_frame Struct Reference

Kernel oops stack frame.

#include <koops\_frame.h>Collaboration diagram for btp\_koops\_frame:



### **Data Fields**

- uint64\_t address
- bool reliable
- char \* function\_name
- uint64\_t **function\_offset**
- uint64\_t function\_length
- char \* module\_name
- uint64\_t from\_address
- char \* from\_function\_name
- uint64 t from function offset
- uint64\_t from\_function\_length
- char \* from\_module\_name
- struct btp\_koops\_frame \* next

# **8.17.1** Detailed Description

Kernel oops stack frame.

### **8.17.2** Field Documentation

### 8.17.2.1 uint64\_t btp\_koops\_frame::address

Address of the function in memory. It is set to 0 when the address is not available. In such a case, function\_name is available.

# 8.17.2.2 uint64\_t btp\_koops\_frame::from\_address

It is set to 0 when the address is not available.

# 8.17.2.3 char\* btp\_koops\_frame::from\_function\_name

Might be NULL.

# $\textbf{8.17.2.4} \quad char*\ btp\_koops\_frame::from\_module\_name$

Might be NULL.

# 8.17.2.5 char\* btp\_koops\_frame::function\_name

Might be NULL. If it is null, address must be set.

# $8.17.2.6 \quad char*\ btp\_koops\_frame::module\_name$

Might be NULL.

# $8.17.2.7 \quad bool\ btp\_koops\_frame::reliable$

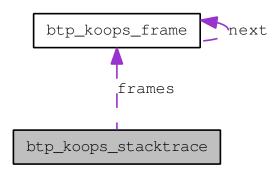
http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git;a=blob;f=arch/x86/kernel/dumpstack.cprintk\_address(unsigned long address, int reliable)

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

• koops\_frame.h

# 8.18 btp\_koops\_stacktrace Struct Reference

Collaboration diagram for btp\_koops\_stacktrace:



# **Data Fields**

• char \* version

Version of the kernel.

- bool taint\_module\_proprietary
- bool taint\_module\_gpl
- bool taint\_module\_out\_of\_tree
- bool taint\_forced\_module
- bool taint\_forced\_removal
- bool taint\_smp\_unsafe
- bool taint\_mce
- bool taint\_page\_release
- bool taint\_userspace
- bool taint\_died\_recently
- bool taint\_acpi\_overridden
- bool taint\_warning
- bool taint\_staging\_driver
- bool taint\_firmware\_workaround
- char \*\* modules

List of loaded modules.

• struct btp\_koops\_frame \* frames

Call trace. It might be NULL as it is not mandatory.

### **8.18.1** Field Documentation

### 8.18.1.1 char\*\* btp\_koops\_stacktrace::modules

List of loaded modules. It might be NULL as it is sometimes not included in a kerneloops.

# 8.18.1.2 bool btp\_koops\_stacktrace::taint\_mce

A machine check exception has been raised.

# 8.18.1.3 bool btp\_koops\_stacktrace::taint\_module\_proprietary

http://www.mjmwired.net/kernel/Documentation/oops-tracing.txt

# $8.18.1.4 \quad bool\ btp\_koops\_stacktrace{::}taint\_page\_release$

A process has been found in a bad page state.

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

• koops\_stacktrace.h

# 8.19 btp\_location Struct Reference

A location of a parser in the input stream.

#include <location.h>

# **Data Fields**

- int line
- int column
- const char \* message

# 8.19.1 Detailed Description

A location of a parser in the input stream. A location in the stacktrace file with an attached message. It's used for error reporting: the line and the column points to the place where a parser error occurred, and the message explains what the parser expected and didn't find on that place.

### **8.19.2** Field Documentation

### 8.19.2.1 int btp\_location::column

Starts from 0.

### 8.19.2.2 int btp\_location::line

Starts from 1.

### 8.19.2.3 const char\* btp\_location::message

Error message related to the line and column. Do not release the memory this pointer points to.

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

• location.h

# 8.20 btp\_operating\_system Struct Reference

# **Data Fields**

- char \* name
- char \* version
- char \* architecture
- uint64\_t **uptime**

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

• report.h

# 8.21 btp\_python\_frame Struct Reference

Collaboration diagram for btp\_python\_frame:



# **Data Fields**

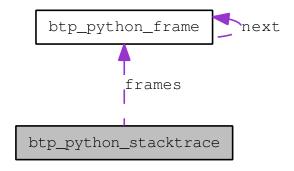
- char \* file\_name
- uint32\_t file\_line
- bool is\_module
- char \* function\_name
- char \* line
- struct btp\_python\_frame \* **next**

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

• python\_frame.h

# 8.22 btp\_python\_stacktrace Struct Reference

Collaboration diagram for btp\_python\_stacktrace:



# **Data Fields**

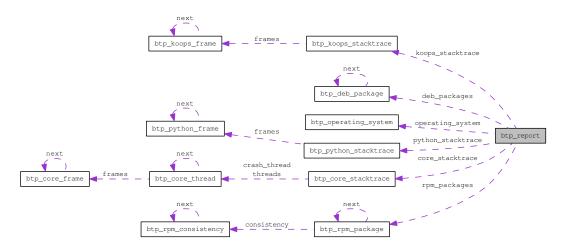
- char \* file\_name
- uint32\_t file\_line
- char \* exception\_name
- struct btp\_python\_frame \* **frames**

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

• python\_stacktrace.h

# 8.23 btp\_report Struct Reference

Collaboration diagram for btp\_report:



# **Data Fields**

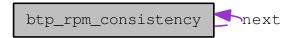
- uint32\_t report\_version
- enum btp\_report\_type report\_type
- char \* reporter\_name
- char \* reporter\_version
- enum btp\_user\_type user\_type
- struct btp\_operating\_system operating\_system
- char \* component\_name
- struct btp\_rpm\_package \* rpm\_packages
- struct btp\_deb\_package \* deb\_packages
- struct btp\_python\_stacktrace \* python\_stacktrace
- struct btp\_koops\_stacktrace \* koops\_stacktrace
- struct btp\_core\_stacktrace \* core\_stacktrace

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

• report.h

# 8.24 btp\_rpm\_consistency Struct Reference

Collaboration diagram for btp\_rpm\_consistency:



# **Data Fields**

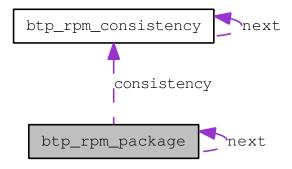
- char \* file\_name
- bool owner\_changed
- bool group\_changed
- bool mode\_changed
- bool md5\_mismatch
- bool size\_changed
- bool major\_number\_changed
- bool minor\_number\_changed
- bool symlink\_changed
- bool modification\_time\_changed
- struct btp\_rpm\_consistency \* next

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

• rpm.h

# 8.25 btp\_rpm\_package Struct Reference

Collaboration diagram for btp\_rpm\_package:



# **Data Fields**

- char \* name
- uint32\_t epoch
- char \* version
- char \* release
- char \* architecture
- uint32\_t install\_time
- struct btp\_rpm\_consistency \* **consistency**
- struct btp\_rpm\_package \* next

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

• rpm.h

# 8.26 btp\_sha1\_state Struct Reference

Internal state of a SHA-1 hash algorithm.

#include <sha1.h>

# **Data Fields**

- uint8\_t wbuffer [64]
- uint64\_t total64
- uint32\_t hash [8]

# 8.26.1 Detailed Description

Internal state of a SHA-1 hash algorithm.

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

• sha1.h

# 8.27 btp\_strbuf Struct Reference

A resizable string buffer.

#include <strbuf.h>

# **Data Fields**

- int alloc
- int len
- char \* buf

# **8.27.1** Detailed Description

A resizable string buffer.

# 8.27.2 Field Documentation

# 8.27.2.1 int btp\_strbuf::alloc

Size of the allocated buffer. Always > 0.

# 8.27.2.2 int btp\_strbuf::len

Length of the string, without the ending.

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

• strbuf.h

# 8.28 btp\_unstrip\_entry Struct Reference

Core dump memory layout as reported by the unstrip utility.

#include <unstrip.h>Collaboration diagram for btp\_unstrip\_entry:



# **Data Fields**

- uint64\_t start
- uint64\_t length
- char \* build\_id
- char \* file\_name
- char \* mod\_name
- struct btp\_unstrip\_entry \* next

# 8.28.1 Detailed Description

Core dump memory layout as reported by the unstrip utility.

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

• unstrip.h

# **Chapter 9**

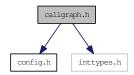
# **File Documentation**

# 9.1 callgraph.h File Reference

Calling relationships between subroutines. #include "config.h"

#include <inttypes.h>

Include dependency graph for callgraph.h:



# **Data Structures**

• struct btp\_callgraph

A call graph representing calling relationships between subroutines.

### **Functions**

- struct btp\_callgraph \* btp\_callgraph\_compute (struct btp\_disasm\_state \*disassembler, struct btp\_elf\_fde \*eh\_frame, char \*\*error\_message)
- struct btp\_callgraph \* btp\_callgraph\_extend (struct btp\_callgraph \*callgraph, uint64\_t start\_address, struct btp\_disasm\_state \*disassembler, struct btp\_elf\_fde \*eh\_frame, char \*\*error\_message)
- void **btp\_callgraph\_free** (struct btp\_callgraph \*callgraph)
- struct btp\_callgraph \* btp\_callgraph\_find (struct btp\_callgraph \*callgraph, uint64\_t address)
- struct btp\_callgraph \* btp\_callgraph\_last (struct btp\_callgraph \*callgraph)

# 9.1.1 Detailed Description

Calling relationships between subroutines. Call graph represents calling relationships between subroutines. In our case, we create the call graph from ELF binaries. Only static relationships obtained from CALL-like instructions with numeric offsets are handled.

File Documentation

Call graph is used by fingerprinting algorithms.

# **9.1.2** Function Documentation

9.1.2.1 struct btp\_callgraph\* btp\_callgraph\_extend (struct btp\_callgraph \* callgraph, uint64\_t start\_address, struct btp\_disasm\_state \* disassembler, struct btp\_elf\_fde \* eh\_frame, char \*\* error\_message) [read]

Assumption: when a fde is included in the callgraph, we assume that all callees are included as well.

# 9.2 cluster.h File Reference

Clustering for stack trace threads.

# **Data Structures**

• struct btp\_dendrogram

A dendrogram created by clustering.

• struct btp\_cluster

A cluster of objects from a dendrogram.

### **Functions**

- struct btp\_dendrogram \* btp\_dendrogram\_new (int size)
- void btp\_dendrogram\_free (struct btp\_dendrogram \*dendrogram)
- struct btp\_dendrogram \* btp\_distances\_cluster\_objects (struct btp\_distances \*distances)
- struct btp\_cluster \* btp\_cluster\_new (int size)
- void btp\_cluster\_free (struct btp\_cluster \*cluster)
- struct btp\_cluster \* btp\_dendrogram\_cut (struct btp\_dendrogram \*dendrogram, float level, int min\_size)

# 9.2.1 Detailed Description

Clustering for stack trace threads. The implemented clustering algorithm assigns a set of stack trace threads into groups. Each group represents a single program flaw.

### 9.2.2 Function Documentation

### **9.2.2.1** void btp\_cluster\_free (struct btp\_cluster \* cluster)

Releases the memory held by the cluster.

#### **Parameters:**

dendrogram If cluster is NULL, no operation is performed.

### 9.2.2.2 struct btp\_cluster\* btp\_cluster\_new (int size) [read]

Creates and initializes a new cluster.

### **Parameters:**

size Number of objects in the cluster.

### **Returns:**

It never returns NULL. The returned pointer must be released by btp\_cluster\_free().

64 File Documentation

# 9.2.2.3 struct btp\_cluster\* btp\_dendrogram\_cut (struct btp\_dendrogram \* dendrogram, float level, int min\_size) [read]

Cuts a dendrogram at specified level.

### **Parameters:**

*dendrogram* The dendrogram which should be cut. The structure is not modified by this call. *level* The cutting level of distance.

min size The minimum size of clusters which should be returned.

### **Returns:**

List of clusters, NULL if empty.

### 9.2.2.4 void btp\_dendrogram\_free (struct btp\_dendrogram \* dendrogram)

Releases the memory held by the dendrogram.

### **Parameters:**

dendrogram If dendrogram is NULL, no operation is performed.

# 9.2.2.5 struct btp\_dendrogram\* btp\_dendrogram\_new (int size) [read]

Creates and initializes a new dendrogram structure.

#### **Parameters:**

size Number of objects.

### **Returns:**

It never returns NULL. The returned pointer must be released by btp\_dendrogram\_free().

# 9.2.2.6 struct btp\_dendrogram\* btp\_distances\_cluster\_objects (struct btp\_distances \* distances) [read]

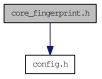
Performs hierarchical agglomerative clustering on objects.

### **Parameters:**

distances Distances between the objects. The structure is not modified by calling this function.

# 9.3 core\_fingerprint.h File Reference

Fingerprint algorithm for core stack traces. #include "config.h" Include dependency graph for core\_fingerprint.h:



# 9.3.1 Detailed Description

Fingerprint algorithm for core stack traces.

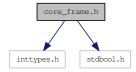
File Documentation

# 9.4 core\_frame.h File Reference

Single frame of core stack trace thread. #include <inttypes.h>

#include <stdbool.h>

Include dependency graph for core\_frame.h:



### **Data Structures**

• struct btp\_core\_frame

A function call on call stack of a core dump.

### **Functions**

- struct btp\_core\_frame \* btp\_core\_frame\_new ()
- void btp\_core\_frame\_init (struct btp\_core\_frame \*frame)
- void btp\_core\_frame\_free (struct btp\_core\_frame \*frame)
- struct btp\_core\_frame \* btp\_core\_frame\_dup (struct btp\_core\_frame \*frame, bool siblings)
- int btp\_core\_frame\_cmp (struct btp\_core\_frame \*frame1, struct btp\_core\_frame \*frame2)
- struct btp\_core\_frame \* btp\_core\_frame\_append (struct btp\_core\_frame \*dest, struct btp\_core\_frame \*item)
- char \* btp\_core\_frame\_to\_json (struct btp\_core\_frame \*frame)

# 9.4.1 Detailed Description

Single frame of core stack trace thread.

### 9.4.2 Function Documentation

9.4.2.1 struct btp\_core\_frame\* btp\_core\_frame\_append (struct btp\_core\_frame \* dest, struct btp\_core\_frame \* item) [read]

Appends 'item' at the end of the list 'dest'.

### **Returns:**

This function returns the 'dest' frame. If 'dest' is NULL, it returns the 'item' frame.

# 9.4.2.2 int btp\_core\_frame\_cmp (struct btp\_core\_frame \* frame1, struct btp\_core\_frame \* frame2)

Compares two frames.

#### **Parameters:**

*frame1* It must be non-NULL pointer. It's not modified by calling this function. *frame2* It must be non-NULL pointer. It's not modified by calling this function.

### **Returns:**

Returns 0 if the frames are same. Returns negative number if frame1 is found to be 'less' than frame2. Returns positive number if frame1 is found to be 'greater' than frame2.

# 9.4.2.3 struct btp\_core\_frame\* btp\_core\_frame\_dup (struct btp\_core\_frame \* frame, bool siblings) [read]

Creates a duplicate of the frame.

#### **Parameters:**

*frame* It must be non-NULL pointer. The frame is not modified by calling this function. *siblings* Whether to duplicate also siblings referenced by frame->next. If false, frame->next is not

siblings Whether to duplicate also siblings referenced by frame->next. If false, frame->next is no duplicated for the new frame, but it is set to NULL.

### **Returns:**

This function never returns NULL. If the returned duplicate is not shallow, it must be released by calling the function btp\_gdb\_frame\_free().

### 9.4.2.4 void btp\_core\_frame\_free (struct btp\_core\_frame \* frame)

Releases the memory held by the frame. The frame siblings are not released.

### **Parameters:**

frame If the frame is NULL, no operation is performed.

### **9.4.2.5** void btp\_core\_frame\_init (struct btp\_core\_frame \* *frame*)

Initializes all members of the frame structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a frame structure placed on the stack.

# 9.4.2.6 struct btp\_core\_frame\* btp\_core\_frame\_new() [read]

Creates and initializes a new frame structure.

#### **Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp\_core\_frame\_free().

File Documentation

# $9.4.2.7 \quad char*\ btp\_core\_frame\_to\_json\ (struct\ btp\_core\_frame*\mathit{frame})$

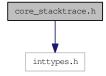
Returns a textual representation of the frame.

# **Parameters:**

frame It must be a non-NULL pointer. It's not modified by calling this function.

# 9.5 core\_stacktrace.h File Reference

A stack trace of a core dump. #include <inttypes.h>
Include dependency graph for core\_stacktrace.h:



### **Data Structures**

• struct btp\_core\_stacktrace

A stack trace of a core dump.

### **Functions**

- struct btp\_core\_stacktrace \* btp\_core\_stacktrace\_new ()
- void btp\_core\_stacktrace\_init (struct btp\_core\_stacktrace \*stacktrace)
- void btp\_core\_stacktrace\_free (struct btp\_core\_stacktrace \*stacktrace)
- struct btp\_core\_stacktrace \* btp\_core\_stacktrace\_dup (struct btp\_core\_stacktrace \*stacktrace)
- int btp\_core\_stacktrace\_get\_thread\_count (struct btp\_core\_stacktrace \*stacktrace)
- struct btp\_core\_stacktrace \* btp\_core\_stacktrace\_parse (const char \*\*input, struct btp\_location \*location)
- char \* btp\_core\_stacktrace\_to\_json (struct btp\_core\_stacktrace \*stacktrace)
- struct btp\_core\_stacktrace \* btp\_core\_stacktrace\_create (const char \*gdb\_stacktrace\_text, const char \*unstrip\_text, const char \*executable\_path)

# 9.5.1 Detailed Description

A stack trace of a core dump.

### 9.5.2 Function Documentation

# 9.5.2.1 struct btp\_core\_stacktrace\* btp\_core\_stacktrace\_dup (struct btp\_core\_stacktrace \* stacktrace) [read]

Creates a duplicate of the stacktrace.

### Parameters:

stacktrace The stacktrace to be copied. It's not modified by this function.

#### **Returns:**

This function never returns NULL. The returned duplicate must be released by calling the function btp\_core\_stacktrace\_free().

70 File Documentation

### 9.5.2.2 void btp\_core\_stacktrace\_free (struct btp\_core\_stacktrace \* stacktrace)

Releases the memory held by the stacktrace, its threads and frames.

#### **Parameters:**

stacktrace If the stacktrace is NULL, no operation is performed.

### 9.5.2.3 int btp\_core\_stacktrace\_get\_thread\_count (struct btp\_core\_stacktrace \* stacktrace)

Returns a number of threads in the stacktrace.

#### **Parameters:**

stacktrace It's not modified by calling this function.

### 9.5.2.4 void btp\_core\_stacktrace\_init (struct btp\_core\_stacktrace \* stacktrace)

Initializes all members of the stacktrace structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a stacktrace structure placed on the stack.

### 9.5.2.5 struct btp\_core\_stacktrace\* btp\_core\_stacktrace\_new() [read]

Creates and initializes a new stacktrace structure.

### **Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp\_core\_stacktrace\_free().

# 9.5.2.6 struct btp\_core\_stacktrace\* btp\_core\_stacktrace\_parse (const char \*\* input, struct btp\_location \* location) [read]

Parses a textual stacktrace and puts it into a structure. If parsing fails, the input parameter is not changed and NULL is returned.

### Note:

Stacktrace can be serialized to string via btp\_core\_stacktrace\_to\_text().

### 9.5.2.7 char\* btp\_core\_stacktrace\_to\_json (struct btp\_core\_stacktrace \* stacktrace)

Serializes stacktrace to string. Newly allocated memory containing the textual representation of the provided stacktrace. Caller should free the memory when it's no longer needed.

# 9.6 core\_thread.h File Reference

Single thread of execution of a core stack trace. #include <stdbool.h> Include dependency graph for core\_thread.h:



### **Data Structures**

struct btp\_core\_thread

A thread of execution on call stack of a core dump.

### **Functions**

- struct btp\_core\_thread \* btp\_core\_thread\_new ()
- void btp\_core\_thread\_init (struct btp\_core\_thread \*thread)
- void btp\_core\_thread\_free (struct btp\_core\_thread \*thread)
- struct btp\_core\_thread \* btp\_core\_thread\_dup (struct btp\_core\_thread \*thread, bool siblings)
- int btp\_core\_thread\_cmp (struct btp\_core\_thread \*thread1, struct btp\_core\_thread \*thread2)
- struct btp\_core\_thread \* btp\_core\_thread\_append (struct btp\_core\_thread \*dest, struct btp\_core\_thread \*item)
- int btp\_core\_thread\_get\_frame\_count (struct btp\_core\_thread \*thread)
- char \* btp\_core\_thread\_to\_json (struct btp\_core\_thread \*thread)

### 9.6.1 Detailed Description

Single thread of execution of a core stack trace.

### 9.6.2 Function Documentation

9.6.2.1 struct btp\_core\_thread\* btp\_core\_thread\_append (struct btp\_core\_thread \* dest, struct btp\_core\_thread \* item) [read]

Appends 'item' at the end of the list 'dest'.

### **Returns:**

This function returns the 'dest' thread. If 'dest' is NULL, it returns the 'item' frame.

72 File Documentation

# 9.6.2.2 int btp\_core\_thread\_cmp (struct btp\_core\_thread \* thread1, struct btp\_core\_thread \* thread2)

Compares two threads. When comparing the threads, it compares also their frames, including the frame numbers.

### **Returns:**

Returns 0 if the threads are same. Returns negative number if t1 is found to be 'less' than t2. Returns positive number if t1 is found to be 'greater' than t2.

# 9.6.2.3 struct btp\_core\_thread\* btp\_core\_thread\_dup (struct btp\_core\_thread \* thread, bool siblings) [read]

Creates a duplicate of the thread.

#### **Parameters:**

thread It must be non-NULL pointer. The thread is not modified by calling this function.

*siblings* Whether to duplicate also siblings referenced by thread->next. If false, thread->next is not duplicated for the new frame, but it is set to NULL.

### 9.6.2.4 void btp\_core\_thread\_free (struct btp\_core\_thread \* thread)

Releases the memory held by the thread. The thread siblings are not released. Thread frames are released.

### **Parameters:**

thread If thread is NULL, no operation is performed.

### 9.6.2.5 int btp\_core\_thread\_get\_frame\_count (struct btp\_core\_thread \* thread)

Returns the number of frames in the thread.

### 9.6.2.6 void btp\_core\_thread\_init (struct btp\_core\_thread \* thread)

Initializes all members of the thread to default values. No memory is released, members are simply overwritten. This is useful for initializing a thread structure placed on the stack.

### 9.6.2.7 struct btp\_core\_thread\* btp\_core\_thread\_new() [read]

Creates and initializes a new frame structure.

### **Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp\_core\_thread\_free().

9.7 deb.h File Reference 73

# 9.7 deb.h File Reference

Deb-related structures and utilities. #include <stdbool.h> Include dependency graph for deb.h:



# **Data Structures**

• struct btp\_deb\_package

# **Functions**

• void btp\_deb\_package\_free (struct btp\_deb\_package \*package, bool recursive)

# 9.7.1 Detailed Description

Deb-related structures and utilities.

# 9.8 disasm.h File Reference

BFD-based function disassembler. #include "config.h"

#include <inttypes.h>

#include <stdbool.h>

Include dependency graph for disasm.h:



## **Functions**

- struct btp\_disasm\_state \* btp\_disasm\_init (const char \*file\_name, char \*\*error\_message)
- void **btp\_disasm\_free** (struct btp\_disasm\_state \*state)
- char \*\* btp\_disasm\_get\_function\_instructions (struct btp\_disasm\_state \*state, uint64\_t start\_offset, uint64\_t size, char \*\*error\_message)
- void **btp\_disasm\_instructions\_free** (char \*\*instructions)
- bool btp\_disasm\_instruction\_is\_one\_of (char \*instruction, const char \*\*mnemonics)
- bool btp\_disasm\_instruction\_present (char \*\*instructions, const char \*\*mnemonics)
- bool btp\_disasm\_instruction\_parse\_single\_address\_operand (char \*instruction, uint64\_t \*dest)
- uint64\_t \* btp\_disasm\_get\_callee\_addresses (char \*\*instructions)

# 9.8.1 Detailed Description

BFD-based function disassembler.

### 9.8.2 Function Documentation

9.8.2.1 char\*\* btp\_disasm\_get\_function\_instructions (struct btp\_disasm\_state \* state, uint64\_t start\_offset, uint64\_t size, char \*\* error\_message)

Disassemble the function starting at 'start\_offset' and taking 'size' bytes, returning a list of (char\*) instructions.

9.9 elves.h File Reference 75

# 9.9 elves.h File Reference

Loading PLT and FDEs from ELF binaries. #include <inttypes.h>
Include dependency graph for elves.h:



### **Data Structures**

- struct btp\_elf\_plt\_entry

  A single item of the Procedure Linkage Table present in ELF binaries.
- struct btp\_elf\_fde

  A single Frame Description Entry of the .eh\_frame section present in ELF binaries.

# **Functions**

- struct btp\_elf\_plt\_entry \* btp\_elf\_get\_procedure\_linkage\_table (const char \*filename, char \*\*error\_message)
- void btp\_elf\_procedure\_linkage\_table\_free (struct btp\_elf\_plt\_entry \*entries)
- struct btp\_elf\_plt\_entry \* btp\_elf\_plt\_find\_for\_address (struct btp\_elf\_plt\_entry \*plt, uint64\_t address)
- struct btp\_elf\_fde \* btp\_elf\_get\_eh\_frame (const char \*filename, char \*\*error\_message)
- void btp\_elf\_eh\_frame\_free (struct btp\_elf\_fde \*entries)
- struct btp\_elf\_fde \* **btp\_elf\_find\_fde\_for\_address** (struct btp\_elf\_fde \*eh\_frame, uint64\_t build\_id\_offset)

# 9.9.1 Detailed Description

Loading PLT and FDEs from ELF binaries. File name elf.h cannot be used due to collision with <elf.h> system include.

# 9.9.2 Function Documentation

# 9.9.2.1 struct btp\_elf\_fde\* btp\_elf\_get\_eh\_frame (const char \* filename, char \*\* error\_message) [read]

Reads the .eh\_frame section from an ELF file.

# Parameters:

error\_message Will be filled by an error message if the function fails (returns NULL). Caller is responsible for calling free() on the string pointer. If function succeeds, the pointer is not touched by the function.

## **Returns:**

Returns a linked list of function ranges (function offset and size) on success. Otherwise NULL.

# 9.9.2.2 struct btp\_elf\_plt\_entry\* btp\_elf\_get\_procedure\_linkage\_table (const char \* filename, char \*\* error\_message) [read]

Reads the Procedure Linkage Table from an ELF file.

## **Parameters:**

error\_message Will be filled by an error message if the function fails (returns NULL). Caller is responsible for calling free() on the string pointer. If function succeeds, the pointer is not touched by the function.

## **Returns:**

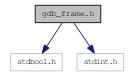
Linked list of PLT entries on success. NULL otherwise.

# 9.10 gdb\_frame.h File Reference

Single frame of GDB stack trace thread. #include <stdbool.h>

#include <stdint.h>

Include dependency graph for gdb\_frame.h:



### **Data Structures**

• struct btp\_gdb\_frame

A function call of a GDB-produced stack trace.

### **Functions**

- struct btp\_gdb\_frame \* btp\_gdb\_frame\_new ()
- void btp\_gdb\_frame\_init (struct btp\_gdb\_frame \*frame)
- void btp\_gdb\_frame\_free (struct btp\_gdb\_frame \*frame)
- struct btp\_gdb\_frame \* btp\_gdb\_frame\_dup (struct btp\_gdb\_frame \*frame, bool siblings)
- bool btp\_gdb\_frame\_calls\_func (struct btp\_gdb\_frame \*frame, const char \*function\_name,...)
- int btp\_gdb\_frame\_cmp (struct btp\_gdb\_frame \*frame1, struct btp\_gdb\_frame \*frame2, bool compare\_number)
- int btp\_gdb\_frame\_cmp\_simple (struct btp\_gdb\_frame \*frame1, struct btp\_gdb\_frame \*frame2)
- struct btp\_gdb\_frame \* btp\_gdb\_frame\_append (struct btp\_gdb\_frame \*dest, struct btp\_gdb\_frame \*item)
- void btp\_gdb\_frame\_append\_to\_str (struct btp\_gdb\_frame \*frame, struct btp\_strbuf \*dest, bool verbose)
- struct btp\_gdb\_frame \* btp\_gdb\_frame\_parse (const char \*\*input, struct btp\_location \*location)
- int btp\_gdb\_frame\_parse\_frame\_start (const char \*\*input, uint32\_t \*number)
- int btp gdb frame parseadd operator (const char \*\*input, struct btp strbuf \*target)
- int btp\_gdb\_frame\_parse\_function\_name\_chunk (const char \*\*input, bool space\_allowed, char \*\*target)
- int btp\_gdb\_frame\_parse\_function\_name\_braces (const char \*\*input, char \*\*target)
- int btp\_gdb\_frame\_parse\_function\_name\_template (const char \*\*input, char \*\*target)
- bool btp\_gdb\_frame\_parse\_function\_name (const char \*\*input, char \*\*function\_name, char \*\*function\_type, struct btp\_location \*location)
- bool btp\_gdb\_frame\_skip\_function\_args (const char \*\*input, struct btp\_location \*location)
- bool btp\_gdb\_frame\_parse\_function\_call (const char \*\*input, char \*\*function\_name, char \*\*function\_type, struct btp\_location \*location)
- bool btp\_gdb\_frame\_parse\_address\_in\_function (const char \*\*input, uint64\_t \*address, char \*\*function\_name, char \*\*function\_type, struct btp\_location \*location)
- bool btp\_gdb\_frame\_parse\_file\_location (const char \*\*input, char \*\*file, uint32\_t \*file\_line, struct btp\_location \*location)

• struct btp\_gdb\_frame \* btp\_gdb\_frame\_parse\_header (const char \*\*input, struct btp\_location \*location)

• void btp\_gdb\_frame\_remove\_func\_prefix (struct btp\_gdb\_frame \*frame, const char \*prefix, int num)

## 9.10.1 Detailed Description

Single frame of GDB stack trace thread.

### 9.10.2 Function Documentation

# 9.10.2.1 struct btp\_gdb\_frame\* btp\_gdb\_frame\_append (struct btp\_gdb\_frame \* dest, struct btp\_gdb\_frame \* item) [read]

Appends 'item' at the end of the list 'dest'.

### **Returns:**

This function returns the 'dest' frame. If 'dest' is NULL, it returns the 'item' frame.

# 9.10.2.2 void btp\_gdb\_frame\_append\_to\_str (struct btp\_gdb\_frame \* frame, struct btp\_strbuf \* dest, bool verbose)

Appends the textual representation of the frame to the string buffer.

## **Parameters:**

frame It must be a non-NULL pointer. It's not modified by calling this function.

# 9.10.2.3 bool btp\_gdb\_frame\_calls\_func (struct btp\_gdb\_frame \* frame, const char \* function\_name, ...)

Checks whether the frame represents a call of function with certain function name.

### **Parameters:**

frame A stack trace frame.

... Names of source files or shared libaries that should contain the function name. The list needs to be terminated by NULL. Just NULL can be provided, and source file cannot be present in order to succeed. An empty string will cause ANY source file to match and succeed. The name of source file is searched as a substring.

### **Returns:**

True if the frame corresponds to a function with function\_name, residing in a source file.

# 9.10.2.4 int btp\_gdb\_frame\_cmp (struct btp\_gdb\_frame \* frame1, struct btp\_gdb\_frame \* frame2, bool compare\_number)

Compares two frames.

### **Parameters:**

*frame1* It must be non-NULL pointer. It's not modified by calling this function.

*frame2* It must be non-NULL pointer. It's not modified by calling this function.

*compare\_number* Indicates whether to include the frame numbers in the comparsion. If set to false, the frame numbers are ignored.

### **Returns:**

Returns 0 if the frames are same. Returns negative number if frame1 is found to be 'less' than frame2. Returns positive number if frame1 is found to be 'greater' than frame2.

# 9.10.2.5 int btp\_gdb\_frame\_cmp\_simple (struct btp\_gdb\_frame \* frame1, struct btp\_gdb\_frame \* frame2)

Compares two frames, but only by their function and library names. Two unknown functions ("??") are assumed to be different and unknown library names to be the same.

### **Parameters:**

frame1 It must be non-NULL pointer. It's not modified by calling this function.

frame2 It must be non-NULL pointer. It's not modified by calling this function.

### **Returns:**

Returns 0 if the frames are same. Returns negative number if frame1 is found to be 'less' than frame2. Returns positive number if frame1 is found to be 'greater' than frame2.

# 9.10.2.6 struct btp\_gdb\_frame\* btp\_gdb\_frame\_dup (struct btp\_gdb\_frame \* frame, bool siblings) [read]

Creates a duplicate of the frame.

### **Parameters:**

frame It must be non-NULL pointer. The frame is not modified by calling this function.

*siblings* Whether to duplicate also siblings referenced by frame->next. If false, frame->next is not duplicated for the new frame, but it is set to NULL.

### **Returns:**

This function never returns NULL. The returned duplicate frame must be released by calling the function btp\_gdb\_frame\_free().

### 9.10.2.7 void btp\_gdb\_frame\_free (struct btp\_gdb\_frame \* frame)

Releases the memory held by the frame. The frame siblings are not released.

### **Parameters:**

frame If the frame is NULL, no operation is performed.

### 9.10.2.8 void btp\_gdb\_frame\_init (struct btp\_gdb\_frame \* frame)

Initializes all members of the frame structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a frame structure placed on the stack.

### 9.10.2.9 struct btp\_gdb\_frame\* btp\_gdb\_frame\_new() [read]

Creates and initializes a new frame structure.

### **Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp\_gdb\_frame\_free().

# 9.10.2.10 struct btp\_gdb\_frame\* btp\_gdb\_frame\_parse (const char \*\* input, struct btp\_location \* location) [read]

If the input contains a complete frame, this function parses the frame text, returns it in a structure, and moves the input pointer after the frame. If the input does not contain proper, complete frame, the function does not modify input and returns NULL.

#### **Returns:**

Allocated pointer with a frame structure. The pointer should be released by btp\_gdb\_frame\_free().

### **Parameters:**

*location* The caller must provide a pointer to an instance of btp\_location here. When this function returns NULL, the structure will contain the error line, column, and message. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values.

# 9.10.2.11 bool btp\_gdb\_frame\_parse\_address\_in\_function (const char \*\* input, uint64\_t \* address, char \*\* function\_name, char \*\* function\_type, struct btp\_location \* location)

If the input contains address and function call, parse them, move the input pointer after this sequence, and return true. Otherwise do not modify the input and return false.

If this function returns true, the caller is responsible to free the parameter function.

```
0x000000322160e7fd in fsync ()
0x000000322222987a in write_to_temp_file (
filename=0x18971b0 "/home/jfclere/.recently-used.xbel",
contents=<value optimized out>, length=29917, error=0x7fff3cbe4110)
```

### **Parameters:**

**location** The caller must provide a pointer to an instance of btp\_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

# 9.10.2.12 bool btp\_gdb\_frame\_parse\_file\_location (const char \*\* input, char \*\* file, uint32\_t \* file\_line, struct btp\_location \* location)

If the input contains sequence "from path/to/file:fileline" or "at path/to/file:fileline", parse it, move the input pointer after this sequence and return true. Otherwise do not modify the input and return false.

The ':' followed by line number is optional. If it is not present, the fileline is set to -1.

#### **Parameters:**

**location** The caller must provide a pointer to an instance of btp\_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

## 9.10.2.13 int btp\_gdb\_frame\_parse\_frame\_start (const char \*\* input, uint32\_t \* number)

If the input contains a proper frame start section, parse the frame number, and move the input pointer after this section. Otherwise do not modify input.

#### **Returns:**

The number of characters parsed from input. 0 if the input does not contain a frame start.

```
"#1 "
"#255 "
```

# 9.10.2.14 bool btp\_gdb\_frame\_parse\_function\_call (const char \*\* input, char \*\* function\_name, char \*\* function\_type, struct btp\_location \* location)

If the input contains proper function call, parse the function name and store it to result, move the input pointer after whole function call, and return true. Otherwise do not modify the input and return false.

If this function returns true, the caller is responsible to free the the function name.

### **Parameters:**

**location** The caller must provide a pointer to an instance of btp\_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

# 9.10.2.15 bool btp\_gdb\_frame\_parse\_function\_name (const char \*\* input, char \*\* function\_name, char \*\* function\_type, struct btp\_location \* location)

Parses the function name, which is a part of the frame header, from the input. If the frame header contains also the function type, it's also parsed.

### **Parameters:**

**function\_name** A pointer pointing to an uninitialized pointer. This function allocates a string and sets the pointer to it if it parses the function name from the input successfully. The memory returned this way must be released by the caller using the function free(). If this function returns true, this pointer is guaranteed to be non-NULL.

**location** The caller must provide a pointer to an instance of btp\_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

### **Returns:**

True if the input stream contained a function name, which has been parsed. False otherwise.

### 9.10.2.16 int btp\_gdb\_frame\_parse\_function\_name\_braces (const char \*\* input, char \*\* target)

If the input buffer contains part of function name containing braces, for example "(anonymous namespace)", parse it, append the contents to target and move input after the braces. Otherwise do not modify the input and the target.

### **Returns:**

The number of characters parsed from input. 0 if the input does not contain a braced part of function name.

# 9.10.2.17 int btp\_gdb\_frame\_parse\_function\_name\_chunk (const char \*\* input, bool space\_allowed, char \*\* target)

Parses a part of function name from the input.

### **Parameters:**

*target* Pointer to a non-allocated pointer. This function will set the pointer to newly allocated memory containing the name chunk, if it returns positive, nonzero value.

### **Returns:**

The number of characters parsed from input. 0 if the input does not contain a part of function name.

# 9.10.2.18 int btp\_gdb\_frame\_parse\_function\_name\_template (const char \*\* input, char \*\* target)

### **Returns:**

The number of characters parsed from input. 0 if the input does not contain a template part of function name.

# 9.10.2.19 struct btp\_gdb\_frame\* btp\_gdb\_frame\_parse\_header (const char \*\* input, struct btp\_location \* location) [read]

If the input contains proper frame header, this function parses the frame header text, moves the input pointer after the frame header, and returns a frame struct. If the input does not contain proper frame header, this function returns NULL and does not modify input.

### **Parameters:**

**location** The caller must provide a pointer to an instance of btp\_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location

should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

### **Returns:**

Newly created frame struct or NULL. The returned frame struct should be released by btp\_gdb\_frame\_free().

## 9.10.2.20 int btp\_gdb\_frame\_parseadd\_operator (const char \*\* input, struct btp\_strbuf \* target)

Parses C++ operator on input. Supports even 'operator new[]' and 'operator delete[]'.

### **Parameters:**

*target* The parsed operator name is appened to the string buffer provided, if an operator is found. Otherwise the string buffer is not changed.

### **Returns:**

The number of characters parsed from input. 0 if the input does not contain operator.

# 9.10.2.21 void btp\_gdb\_frame\_remove\_func\_prefix (struct btp\_gdb\_frame \* frame, const char \* prefix, int num)

Removes first num chars from function name in the frame if it begins with the prefix.

# 9.10.2.22 bool btp\_gdb\_frame\_skip\_function\_args (const char \*\* input, struct btp\_location \* location)

Skips function arguments which are a part of the frame header, in the input stream.

### **Parameters:**

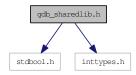
**location** The caller must provide a pointer to an instance of btp\_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

# 9.11 gdb\_sharedlib.h File Reference

Shared library information as produced by GDB. #include <stdbool.h>

#include <inttypes.h>

Include dependency graph for gdb\_sharedlib.h:



### **Data Structures**

• struct btp\_gdb\_sharedlib

A shared library memory location as reported by GDB.

## **Enumerations**

• enum { SYMS\_OK, SYMS\_WRONG, SYMS\_NOT\_FOUND }

## **Functions**

- struct btp\_gdb\_sharedlib \* btp\_gdb\_sharedlib\_new ()
- void btp\_gdb\_sharedlib\_init (struct btp\_gdb\_sharedlib \*sharedlib)
- void btp\_gdb\_sharedlib\_free (struct btp\_gdb\_sharedlib \*sharedlib)
- struct btp\_gdb\_sharedlib \* btp\_gdb\_sharedlib\_append (struct btp\_gdb\_sharedlib \*dest, struct btp\_gdb\_sharedlib \*item)
- struct btp\_gdb\_sharedlib \* btp\_gdb\_sharedlib\_dup (struct btp\_gdb\_sharedlib \*sharedlib, bool sib-lings)
- int btp\_gdb\_sharedlib\_count (struct btp\_gdb\_sharedlib \*sharedlib)
- struct btp\_gdb\_sharedlib \* btp\_gdb\_sharedlib\_find\_address (struct btp\_gdb\_sharedlib \*first, uint64\_t address)
- struct btp\_gdb\_sharedlib \* btp\_gdb\_sharedlib\_parse (const char \*input)

# 9.11.1 Detailed Description

Shared library information as produced by GDB.

## 9.11.2 Function Documentation

9.11.2.1 struct btp\_gdb\_sharedlib\* btp\_gdb\_sharedlib\_append (struct btp\_gdb\_sharedlib\* dest, struct btp\_gdb\_sharedlib\* item) [read]

Appends 'item' at the end of the list 'dest'.

### **Returns:**

This function returns the 'dest' sharedlib. If 'dest' is NULL, it returns the 'item' sharedlib.

### 9.11.2.2 int btp gdb sharedlib count (struct btp gdb sharedlib \* sharedlib)

Returns the number of sharedlibs in the list.

# 9.11.2.3 struct btp\_gdb\_sharedlib\* btp\_gdb\_sharedlib\_dup (struct btp\_gdb\_sharedlib\* sharedlib\*, bool siblings) [read]

Creates a duplicate of the sharedlib structure.

### **Parameters:**

sharedlib Structure to be duplicated.siblings Whether to duplicate a single structure or whole list.

### **Returns:**

Never returns NULL. Returns the duplicated structure or the first structure in the duplicated list.

# 9.11.2.4 struct btp\_gdb\_sharedlib\* btp\_gdb\_sharedlib\_find\_address (struct btp\_gdb\_sharedlib \* first, uint64\_t address) [read]

Finds whether the address belongs to some sharedlib from the list starting by 'first'.

# **Returns:**

Pointer to an existing structure or NULL if not found.

# 9.11.2.5 void btp\_gdb\_sharedlib\_free (struct btp\_gdb\_sharedlib \* sharedlib)

Releases the memory held by the sharedlib. Sharedlibs referenced by .next are not released.

### **Parameters:**

sharedlib If sharedlib is NULL, no operation is performed.

## 9.11.2.6 void btp\_gdb\_sharedlib\_init (struct btp\_gdb\_sharedlib \* sharedlib)

Initializes all members of the sharedlib to default values. No memory is released, members are simply overwritten. This is useful for initializing a sharedlib structure placed on the stack.

### 9.11.2.7 struct btp gdb sharedlib\* btp gdb sharedlib new () [read]

Creates and initializes a new sharedlib structure.

### **Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp\_gdb\_sharedlib\_free().

# 9.11.2.8 struct btp\_gdb\_sharedlib\* btp\_gdb\_sharedlib\_parse (const char \* input) [read]

Parses the output of GDB's 'info sharedlib' command.

# **Parameters:**

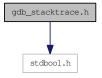
input String representing the stacktrace.

# **Returns:**

First element of the list of loaded libraries.

# 9.12 gdb\_stacktrace.h File Reference

Stack trace as produced by GDB. #include <stdbool.h>
Include dependency graph for gdb\_stacktrace.h:



### **Data Structures**

• struct btp\_gdb\_stacktrace

A stack trace produced by GDB.

### **Functions**

- struct btp\_gdb\_stacktrace \* btp\_gdb\_stacktrace\_new ()
- void btp\_gdb\_stacktrace\_init (struct btp\_gdb\_stacktrace \*stacktrace)
- void btp\_gdb\_stacktrace\_free (struct btp\_gdb\_stacktrace \*stacktrace)
- struct btp\_gdb\_stacktrace \* btp\_gdb\_stacktrace dup (struct btp\_gdb\_stacktrace \*stacktrace)
- int btp\_gdb\_stacktrace\_get\_thread\_count (struct btp\_gdb\_stacktrace \*stacktrace)
- void btp\_gdb\_stacktrace\_remove\_threads\_except\_one (struct btp\_gdb\_stacktrace \*stacktrace, struct btp\_gdb\_thread \*thread)
- struct btp\_gdb\_thread \* btp\_gdb\_stacktrace\_find\_crash\_thread (struct btp\_gdb\_stacktrace \*stacktrace)
- void btp\_gdb\_stacktrace\_limit\_frame\_depth (struct btp\_gdb\_stacktrace \*stacktrace, int depth)
- float btp\_gdb\_stacktrace\_quality\_simple (struct btp\_gdb\_stacktrace \*stacktrace)
- float btp\_gdb\_stacktrace\_quality\_complex (struct btp\_gdb\_stacktrace \*stacktrace)
- char \* btp\_gdb\_stacktrace\_to\_text (struct btp\_gdb\_stacktrace \*stacktrace, bool verbose)
- struct btp\_gdb\_frame \* btp\_gdb\_stacktrace\_get\_crash\_frame (struct btp\_gdb\_stacktrace \*stacktrace)
- char \* btp\_gdb\_stacktrace\_get\_duplication\_hash (struct btp\_gdb\_stacktrace \*stacktrace)
- struct btp\_gdb\_stacktrace \* btp\_gdb\_stacktrace\_parse (const char \*\*input, struct btp\_location \*location)
- bool btp\_gdb\_stacktrace\_parse\_header (const char \*\*input, struct btp\_gdb\_frame \*\*frame, struct btp location \*location)
- void btp\_gdb\_stacktrace\_set\_libnames (struct btp\_gdb\_stacktrace \*stacktrace)
- struct btp\_gdb\_thread \* btp\_gdb\_stacktrace\_get\_optimized\_thread (struct btp\_gdb\_stacktrace \*stacktrace, int max\_frames)

# 9.12.1 Detailed Description

Stack trace as produced by GDB.

## 9.12.2 Function Documentation

# 9.12.2.1 struct btp\_gdb\_stacktrace\* btp\_gdb\_stacktrace\_dup (struct btp\_gdb\_stacktrace \* stacktrace) [read]

Creates a duplicate of a stacktrace.

### **Parameters:**

stacktrace The stacktrace to be copied. It's not modified by this function.

### **Returns:**

This function never returns NULL. The returned duplicate must be released by calling the function btp\_gdb\_stacktrace\_free().

# 9.12.2.2 struct btp\_gdb\_thread\* btp\_gdb\_stacktrace\_find\_crash\_thread (struct btp\_gdb\_stacktrace \* stacktrace) [read]

Searches all threads and tries to find the one that caused the crash. It might return NULL if the thread cannot be determined.

### **Parameters:**

stacktrace It must be non-NULL pointer. It's not modified by calling this function.

## 9.12.2.3 void btp\_gdb\_stacktrace\_free (struct btp\_gdb\_stacktrace \* stacktrace)

Releases the memory held by the stacktrace, its threads, frames, shared libraries.

## Parameters:

stacktrace If the stacktrace is NULL, no operation is performed.

# 9.12.2.4 struct btp\_gdb\_frame\* btp\_gdb\_stacktrace\_get\_crash\_frame (struct btp\_gdb\_stacktrace \* stacktrace) [read]

Analyzes the stacktrace to get the frame where a crash occurred.

## **Parameters:**

stacktrace It must be non-NULL pointer. It's not modified by calling this function.

### **Returns:**

The returned value must be released by calling btp\_gdb\_frame\_free() when it's no longer needed, because it is a deep copy of the crash frame from the stacktrace. NULL is returned if the crash frame is not found.

## 9.12.2.5 char\* btp\_gdb\_stacktrace\_get\_duplication\_hash (struct btp\_gdb\_stacktrace \* stacktrace)

Calculates the duplication hash string of the stacktrace.

### **Parameters:**

stacktrace It must be non-NULL pointer. It's not modified by calling this function.

### **Returns:**

This function never returns NULL. The caller is responsible for releasing the returned memory using function free().

# 9.12.2.6 struct btp\_gdb\_thread\* btp\_gdb\_stacktrace\_get\_optimized\_thread (struct btp\_gdb\_stacktrace \* stacktrace, int max frames) [read]

Return crash thread optimized for comparison. It's normalized, with library names set and functions without names (signal handlers) are removed.

### **Parameters:**

stacktrace It must be non-NULL pointer. It's not modified by calling this function.

*max\_frames* The maximum number of frames in the returned crash thread. Superfluous frames are removed from the returned thread.

### **Returns:**

A newly allocated thread structure or NULL. NULL is returned when the crashing thread could not be found. The returned structure should be released by btp\_gdb\_thread\_free() by the caller.

### 9.12.2.7 int btp\_gdb\_stacktrace\_get\_thread\_count (struct btp\_gdb\_stacktrace \* stacktrace)

Returns a number of threads in the stacktrace.

## **Parameters:**

stacktrace It's not modified by calling this function.

### 9.12.2.8 void btp\_gdb\_stacktrace\_init (struct btp\_gdb\_stacktrace \* stacktrace)

Initializes all members of the stacktrace structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a stacktrace structure placed on the stack.

# 9.12.2.9 void btp\_gdb\_stacktrace\_limit\_frame\_depth (struct btp\_gdb\_stacktrace \* stacktrace, int depth)

Remove frames from the bottom of threads in the stacktrace, until all threads have at most 'depth' frames.

### **Parameters:**

stacktrace Must be non-NULL pointer.

### 9.12.2.10 struct btp\_gdb\_stacktrace\* btp\_gdb\_stacktrace\_new() [read]

Creates and initializes a new stack trace structure.

#### **Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp\_gdb\_stacktrace free().

# 9.12.2.11 struct btp\_gdb\_stacktrace\* btp\_gdb\_stacktrace\_parse (const char \*\* input, struct btp\_location \* location) [read]

Parses a textual stack trace and puts it into a structure. If parsing fails, the input parameter is not changed and NULL is returned.

### **Parameters:**

*input* Pointer to the string with the stacktrace. If this function returns a non-NULL value, this pointer is modified to point after the stacktrace that was just parsed.

location The caller must provide a pointer to an instance of btp\_location here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized by btp\_location\_init() before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

## **Returns:**

A newly allocated stacktrace structure or NULL. A stacktrace struct is returned when at least one thread was parsed from the input and no error occurred. The returned structure should be released by btp\_gdb\_stacktrace\_free().

# 9.12.2.12 bool btp\_gdb\_stacktrace\_parse\_header (const char \*\* input, struct btp\_gdb\_frame \*\* frame, struct btp\_location \* location)

Parse stacktrace header if it is available in the stacktrace. The header usually contains frame where the program crashed.

### **Parameters:**

*input* Pointer that will be moved to point behind the header if the header is successfully detected and parsed.

*frame* If this function succeeds and returns true, \*frame contains the crash frame that is usually a part of the header. If no frame is detected in the header, \*frame is set to NULL.

### 9.12.2.13 float btp\_gdb\_stacktrace\_quality\_complex (struct btp\_gdb\_stacktrace \* stacktrace)

Evaluates the quality of the stacktrace. The quality is determined depending on the ratio of frames with function name fully known to all frames.

#### **Parameters:**

stacktrace It must be non-NULL pointer. It's not modified by calling this function.

### **Returns:**

A number between 0 and 1. 0 means the lowest quality, 1 means full stacktrace is known. The returned value takes into account that the thread which caused the crash is more important than the other threads, and the frames around the crash frame are more important than distant frames.

### 9.12.2.14 float btp\_gdb\_stacktrace\_quality\_simple (struct btp\_gdb\_stacktrace \* stacktrace)

Evaluates the quality of the stacktrace. The quality is the ratio of the number of frames with function name fully known to the number of all frames. This function does not take into account that some frames are more important than others.

### **Parameters:**

stacktrace It must be non-NULL pointer. It's not modified by calling this function.

### **Returns:**

A number between 0 and 1. 0 means the lowest quality, 1 means full stacktrace is known (all function names are known).

# 9.12.2.15 void btp\_gdb\_stacktrace\_remove\_threads\_except\_one (struct btp\_gdb\_stacktrace \* stacktrace, struct btp\_gdb\_thread \* thread)

Removes all threads from the stacktrace and deletes them, except the one provided as a parameter.

# **Parameters:**

*thread* This function does not check whether the thread is a member of the stacktrace. If it's not, all threads are removed from the stacktrace and then deleted.

# 9.12.2.16 void btp\_gdb\_stacktrace\_set\_libnames (struct btp\_gdb\_stacktrace \* stacktrace)

Set library names in all frames in the stacktrace according to the the sharedlib data.

# 9.12.2.17 char\* btp\_gdb\_stacktrace\_to\_text (struct btp\_gdb\_stacktrace \* stacktrace, bool verbose)

Returns textual representation of the stacktrace.

## **Parameters:**

stacktrace It must be non-NULL pointer. It's not modified by calling this function.

# **Returns:**

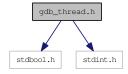
This function never returns NULL. The caller is responsible for releasing the returned memory using function free().

# 9.13 gdb\_thread.h File Reference

Single thread of execution of GDB stack trace. #include <stdbool.h>

#include <stdint.h>

Include dependency graph for gdb\_thread.h:



### **Data Structures**

• struct btp\_gdb\_thread

A thread of execution of a GDB-produced stack trace.

### **Functions**

- struct btp\_gdb\_thread \* btp\_gdb\_thread\_new ()
- void btp\_gdb\_thread\_init (struct btp\_gdb\_thread \*thread)
- void btp\_gdb\_thread\_free (struct btp\_gdb\_thread \*thread)
- struct btp\_gdb\_thread \* btp\_gdb\_thread\_dup (struct btp\_gdb\_thread \*thread, bool siblings)
- int btp\_gdb\_thread\_cmp (struct btp\_gdb\_thread \*thread1, struct btp\_gdb\_thread \*thread2)
- struct btp\_gdb\_thread \* btp\_gdb\_thread\_append (struct btp\_gdb\_thread \*dest, struct btp\_gdb\_thread \*item)
- int btp\_gdb\_thread\_get\_frame\_count (struct btp\_gdb\_thread \*thread)
- void btp\_gdb\_thread\_quality\_counts (struct btp\_gdb\_thread \*thread, int \*ok\_count, int \*all\_count)
- float btp\_gdb\_thread\_quality (struct btp\_gdb\_thread \*thread)
- bool btp\_gdb\_thread\_remove\_frame (struct btp\_gdb\_thread \*thread, struct btp\_gdb\_frame \*frame)
- bool btp\_gdb\_thread\_remove\_frames\_above (struct btp\_gdb\_thread \*thread, struct btp\_gdb\_frame \*frame)
- void btp\_gdb\_thread\_remove\_frames\_below\_n (struct btp\_gdb\_thread \*thread, int n)
- void btp\_gdb\_thread\_append\_to\_str (struct btp\_gdb\_thread \*thread, struct btp\_strbuf \*dest, bool verbose)
- struct btp\_gdb\_thread \* btp\_gdb\_thread\_parse (const char \*\*input, struct btp\_location \*location)
- int btp\_gdb\_thread\_skip\_lwp (const char \*\*input)
- struct btp\_gdb\_thread \* btp\_gdb\_thread\_parse\_funs (const char \*input)
- char \* btp\_gdb\_thread\_format\_funs (struct btp\_gdb\_thread \*thread)

# 9.13.1 Detailed Description

Single thread of execution of GDB stack trace.

## 9.13.2 Function Documentation

# 9.13.2.1 struct btp\_gdb\_thread\* btp\_gdb\_thread\_append (struct btp\_gdb\_thread \* dest, struct btp\_gdb\_thread \* item) [read]

Appends 'item' at the end of the list 'dest'.

#### **Returns:**

This function returns the 'dest' thread.

# 9.13.2.2 void btp\_gdb\_thread\_append\_to\_str (struct btp\_gdb\_thread \* thread, struct btp\_strbuf \* dest, bool verbose)

Appends a textual representation of 'thread' to the 'str'.

# 9.13.2.3 int btp\_gdb\_thread\_cmp (struct btp\_gdb\_thread \* thread1, struct btp\_gdb\_thread \* thread2)

Compares two threads. When comparing the threads, it compares also their frames, including the frame numbers.

### **Returns:**

Returns 0 if the threads are same. Returns negative number if t1 is found to be 'less' than t2. Returns positive number if t1 is found to be 'greater' than t2.

# 9.13.2.4 struct btp\_gdb\_thread\* btp\_gdb\_thread\_dup (struct btp\_gdb\_thread \* thread, bool siblings) [read]

Creates a duplicate of the thread.

### **Parameters:**

thread It must be non-NULL pointer. The thread is not modified by calling this function.

*siblings* Whether to duplicate also siblings referenced by thread->next. If false, thread->next is not duplicated for the new frame, but it is set to NULL.

## 9.13.2.5 char\* btp\_gdb\_thread\_format\_funs (struct btp\_gdb\_thread \* thread)

Prepare a string representing thread which contains just the function and library names. This can be used to store only data necessary for comparison.

# Returns:

Newly allocated string, which should be released by calling free(). The string can be parsed by btp\_gdb\_thread\_parse\_funs().

## 9.13.2.6 void btp\_gdb\_thread\_free (struct btp\_gdb\_thread \* thread)

Releases the memory held by the thread. The thread siblings are not released.

#### **Parameters:**

thread If thread is NULL, no operation is performed.

### 9.13.2.7 int btp\_gdb\_thread\_get\_frame\_count (struct btp\_gdb\_thread \* thread)

Returns the number of frames in the thread.

### 9.13.2.8 void btp\_gdb\_thread\_init (struct btp\_gdb\_thread \* thread)

Initializes all members of the thread to default values. No memory is released, members are simply overwritten. This is useful for initializing a thread structure placed on the stack.

### 9.13.2.9 struct btp\_gdb\_thread\* btp\_gdb\_thread\_new() [read]

Creates and initializes a new frame structure.

#### Returns:

It never returns NULL. The returned pointer must be released by calling the function btp\_gdb\_thread\_free().

# 9.13.2.10 struct btp\_gdb\_thread\* btp\_gdb\_thread\_parse (const char \*\* input, struct btp\_location \* location) [read]

If the input contains proper thread with frames, parse the thread, move the input pointer after the thread, and return a structure representing the thread. Otherwise to not modify the input pointer and return NULL.

## **Parameters:**

*location* The caller must provide a pointer to struct btp\_location here. The line and column members are gradually increased as the parser handles the input, keep this in mind to get reasonable values. When this function returns NULL (an error occurred), the structure will contain the error line, column, and message.

### **Returns:**

NULL or newly allocated structure, which should be released by calling btp\_gdb\_thread\_free().

## 9.13.2.11 struct btp\_gdb\_thread\* btp\_gdb\_thread\_parse\_funs (const char \* input) [read]

Create a thread from function and library names.

### **Parameters:**

input String containing function names and library names separated by space, one frame per line.

### **Returns:**

Newly allocated structure, which should be released by calling btp\_gdb\_thread\_free().

## 9.13.2.12 float btp\_gdb\_thread\_quality (struct btp\_gdb\_thread \* thread)

Returns the quality of the thread. The quality is the ratio of the number of frames with function name fully known to the number of all frames. This function does not take into account that some frames are more important than others.

#### **Parameters:**

thread Must be a non-NULL pointer. It's not modified in this function.

### **Returns:**

A number between 0 and 1. 0 means the lowest quality, 1 means full thread stacktrace is known. If the thread contains no frames, this function returns 1.

# 9.13.2.13 void btp\_gdb\_thread\_quality\_counts (struct btp\_gdb\_thread \* thread, int \* ok\_count, int \* all count)

Counts the number of 'good' frames and the number of all frames in a thread. Good means that the function name is known (so it's not just '??').

### **Parameters:**

ok count

all\_count Not zeroed. This function just adds the numbers to ok\_count and all\_count.

# 9.13.2.14 bool btp\_gdb\_thread\_remove\_frame (struct btp\_gdb\_thread \* thread, struct btp\_gdb\_frame \* frame)

Removes the frame from the thread and then deletes it.

### **Returns:**

True if the frame was found in the thread and removed and deleted. False if the frame was not found in the thread.

# 9.13.2.15 bool btp\_gdb\_thread\_remove\_frames\_above (struct btp\_gdb\_thread \* thread, struct btp\_gdb\_frame \* frame)

Removes all the frames from the thread that are above certain frame.

# Returns:

True if the frame was found, and all the frames that were above the frame in the thread were removed from the thread and then deleted. False if the frame was not found in the thread.

### 9.13.2.16 void btp\_gdb\_thread\_remove\_frames\_below\_n (struct btp\_gdb\_thread \* thread, int n)

Keeps only the top n frames in the thread.

# 9.13.2.17 int btp\_gdb\_thread\_skip\_lwp (const char \*\* input)

If the input contains a LWP section in form of (LWP [0-9]+), move the input pointer after this section. Otherwise do not modify input.

## **Returns:**

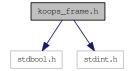
The number of characters parsed from input. 0 if the input does not contain a LWP section.

# 9.14 koops\_frame.h File Reference

Kernel oops stack frame. #include <stdbool.h>

#include <stdint.h>

Include dependency graph for koops\_frame.h:



### **Data Structures**

• struct btp\_koops\_frame

Kernel oops stack frame.

## **Functions**

- struct btp\_koops\_frame \* btp\_koops\_frame\_new ()
- void btp\_koops\_frame\_init (struct btp\_koops\_frame \*frame)
- void btp\_koops\_frame\_free (struct btp\_koops\_frame \*frame)
- struct btp\_koops\_frame \* btp\_koops\_frame\_dup (struct btp\_koops\_frame \*frame, bool siblings)
- int btp\_koops\_frame\_cmp (struct btp\_koops\_frame \*frame1, struct btp\_koops\_frame \*frame2)
- struct btp\_koops\_frame \* btp\_koops\_frame\_append (struct btp\_koops\_frame \*dest, struct btp\_koops\_frame \*item)
- struct btp\_koops\_frame \* btp\_koops\_frame\_parse (const char \*\*input)
- bool btp\_koops\_skip\_timestamp (const char \*\*input)
- bool btp\_koops\_parse\_address (const char \*\*input, uint64\_t \*address)
- bool btp\_koops\_parse\_module\_name (const char \*\*input, char \*\*module\_name)
- bool **btp\_koops\_parse\_function** (const char \*\*input, char \*\*function\_name, uint64\_t \*function\_offset, uint64\_t \*function\_length, char \*\*module\_name)
- char \* btp\_koops\_frame\_to\_json (struct btp\_koops\_frame \*frame)

# 9.14.1 Detailed Description

Kernel oops stack frame.

## 9.14.2 Function Documentation

9.14.2.1 struct btp\_koops\_frame\* btp\_koops\_frame\_append (struct btp\_koops\_frame \* dest, struct btp\_koops\_frame \* item) [read]

Appends 'item' at the end of the list 'dest'.

### **Returns:**

This function returns the 'dest' frame. If 'dest' is NULL, it returns the 'item' frame.

# 9.14.2.2 int btp\_koops\_frame\_cmp (struct btp\_koops\_frame \* frame1, struct btp\_koops\_frame \* frame2)

Compares two frames.

#### **Parameters:**

*frame1* It must be non-NULL pointer. It's not modified by calling this function. *frame2* It must be non-NULL pointer. It's not modified by calling this function.

### **Returns:**

Returns 0 if the frames are same. Returns negative number if frame1 is found to be 'less' than frame2. Returns positive number if frame1 is found to be 'greater' than frame2.

# 9.14.2.3 struct btp\_koops\_frame\* btp\_koops\_frame\_dup (struct btp\_koops\_frame \* frame, bool siblings) [read]

Creates a duplicate of the frame.

### **Parameters:**

frame It must be non-NULL pointer. The frame is not modified by calling this function.

*siblings* Whether to duplicate also siblings referenced by frame->next. If false, frame->next is not duplicated for the new frame, but it is set to NULL.

### **Returns:**

This function never returns NULL. The returned duplicate frame must be released by calling the function btp\_koops\_frame\_free().

### 9.14.2.4 void btp\_koops\_frame\_free (struct btp\_koops\_frame \* frame)

Releases the memory held by the frame. The frame siblings are not released.

## Parameters:

frame If the frame is NULL, no operation is performed.

### 9.14.2.5 void btp\_koops\_frame\_init (struct btp\_koops\_frame \* frame)

Initializes all members of the frame structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a frame structure placed on the stack.

## 9.14.2.6 struct btp\_koops\_frame\* btp\_koops\_frame\_new() [read]

Creates and initializes a new frame structure.

### **Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp\_koops\_frame\_free().

# 9.14.2.7 char\* btp\_koops\_frame\_to\_json (struct btp\_koops\_frame \* frame)

Returns a textual representation of the frame.

# **Parameters:**

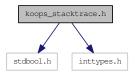
frame It must be a non-NULL pointer. It's not modified by calling this function.

# 9.15 koops\_stacktrace.h File Reference

Kernel oops stack trace structure and related algorithms. #include <stdbool.h>

#include <inttypes.h>

Include dependency graph for koops\_stacktrace.h:



## **Data Structures**

• struct btp\_koops\_stacktrace

## **Functions**

- struct btp\_koops\_stacktrace \* btp\_koops\_stacktrace\_new ()
- void btp\_koops\_stacktrace\_init (struct btp\_koops\_stacktrace \*stacktrace)
- void btp\_koops\_stacktrace\_free (struct btp\_koops\_stacktrace \*stacktrace)
- struct btp\_koops\_stacktrace \* btp\_koops\_stacktrace\_dup (struct btp\_koops\_stacktrace \*stacktrace)
- int btp\_koops\_stacktrace\_get\_frame\_count (struct btp\_koops\_stacktrace \*stacktrace)
- bool btp\_koops\_stacktrace\_remove\_frame (struct btp\_koops\_stacktrace \*stacktrace, struct btp\_koops\_frame \*frame)
- struct btp\_koops\_stacktrace \* btp\_koops\_stacktrace\_parse (const char \*\*input, struct btp\_location \*location)
- char \*\* btp\_koops\_stacktrace\_parse\_modules (const char \*\*input)

## 9.15.1 Detailed Description

Kernel oops stack trace structure and related algorithms.

# 9.15.2 Function Documentation

# 9.15.2.1 struct btp\_koops\_stacktrace\* btp\_koops\_stacktrace\_dup (struct btp\_koops\_stacktrace \* stacktrace) [read]

Creates a duplicate of a stacktrace.

## **Parameters:**

stacktrace The stacktrace to be copied. It's not modified by this function.

## **Returns:**

This function never returns NULL. The returned duplicate must be released by calling the function btp\_koops\_stacktrace\_free().

## 9.15.2.2 void btp\_koops\_stacktrace\_free (struct btp\_koops\_stacktrace \* stacktrace)

Releases the memory held by the stacktrace.

### **Parameters:**

stacktrace If the stacktrace is NULL, no operation is performed.

## 9.15.2.3 int btp\_koops\_stacktrace\_get\_frame\_count (struct btp\_koops\_stacktrace \* stacktrace)

Returns the number of frames in the Kerneloops stacktrace.

### 9.15.2.4 void btp\_koops\_stacktrace\_init (struct btp\_koops\_stacktrace \* stacktrace)

Initializes all members of the stacktrace structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a stacktrace structure placed on the stack.

## 9.15.2.5 struct btp\_koops\_stacktrace\* btp\_koops\_stacktrace\_new() [read]

Creates and initializes a new stack trace structure.

### **Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp\_koops\_stacktrace free().

# 9.15.2.6 struct btp\_koops\_stacktrace\* btp\_koops\_stacktrace\_parse (const char \*\* input, struct btp\_location \* location) [read]

Parses a textual kernel oops and puts it into a structure. If parsing fails, the input parameter is not changed and NULL is returned.

## **Parameters:**

*input* Pointer to the string with the kernel oops. If this function returns a non-NULL value, the input pointer is modified to point after the stacktrace that was just parsed.

# 9.15.2.7 bool btp\_koops\_stacktrace\_remove\_frame (struct btp\_koops\_stacktrace \* stacktrace, struct btp\_koops\_frame \* frame)

Removes the frame from the stack trace and then deletes it.

## **Returns:**

True if the frame was found in the thread and removed and deleted. False if the frame was not found in the thread.

# 9.16 location.h File Reference

Parser location in input file. #include <stdbool.h>

Include dependency graph for location.h:



### **Data Structures**

• struct btp\_location

A location of a parser in the input stream.

### **Functions**

- void btp\_location\_init (struct btp\_location \*location)
- int btp\_location\_cmp (struct btp\_location \*location1, struct btp\_location \*location2, bool compare\_messages)
- char \* btp\_location\_to\_string (struct btp\_location \*location)
- void btp\_location\_add (struct btp\_location \*location, int add\_line, int add\_column)
- void btp\_location\_add\_ext (int \*line, int \*column, int add\_line, int add\_column)
- void btp\_location\_eat\_char (struct btp\_location \*location, char c)
- void btp\_location\_eat\_char\_ext (int \*line, int \*column, char c)

# 9.16.1 Detailed Description

Parser location in input file.

## 9.16.2 Function Documentation

### 9.16.2.1 void btp\_location\_add (struct btp\_location \* location, int add\_line, int add\_column)

Adds a line and a column to specific location.

# Note:

If the line is not 1 (meaning the first line), the column in the location structure is overwritten by the provided add\_column value. Otherwise the add\_column value is added to the column member of the location structure.

### Parameters:

location The structure to be modified. It must be a valid pointer.

add\_line Starts from 1. It means that if add\_line is 1, the line member of the location structure is not changed.

add\_column Starts from 0.

### 9.16.2.2 void btp\_location\_add\_ext (int \* line, int \* column, int add\_line, int add\_column)

Adds a line column pair to another line column pair.

### Note:

If the add\_line is not 1 (meaning the frist line), the column is overwritten by the provided add\_column value. Otherwise the add\_column value is added to the column.

#### Parameters:

```
add_line Starts from 1. It means that if add_line is 1, the line is not changed. add column Starts from 0.
```

# 9.16.2.3 int btp\_location\_cmp (struct btp\_location \* location1, struct btp\_location \* location2, bool compare\_messages)

Compare two locations.

#### Parameters:

```
location1 It must be non-NULL pointer. It's not modified by calling this function.location2 It must be non-NULL pointer. It's not modified by calling this function.compare_messages Indicates whether to compare messages in the locations as well.
```

### **Returns:**

Returns 0 if the locations are same. Returns negative number if location1 is found to be 'less' than location2. Returns positive number if location1 is found to be 'greater' than location2.

'Less' and 'greater' take lines into account first. If a location1 line is lower than location2 line, location1 is considered 'less' than location2. If the lines are the same, columns are compared. When compare\_messages is true and lines and columns are equal, the locations' messages are compared according to the lexicographical order.

## 9.16.2.4 void btp\_location\_eat\_char (struct btp\_location \* location, char c)

Updates the line and column of the location by moving "after" the char c. If c is a newline character, the line number is increased and the column is set to 0. Otherwise the column is increased by 1.

# 9.16.2.5 void btp\_location\_eat\_char\_ext (int \* line, int \* column, char c)

Updates the line and the column by moving "after" the char c. If c is a newline character, the line number is increased and the column is set to 0. Otherwise the column is increased.

### **Parameters:**

```
line Must be a valid pointer.column Must be a valid pointer.
```

# **9.16.2.6** void btp\_location\_init (struct btp\_location \* location)

Initializes all members of the location struct to their default values. No memory is allocated or released by this function.

# 9.16.2.7 char\* btp\_location\_to\_string (struct btp\_location \* location)

Creates a string representation of location. User must delete the returned string using free().

# 9.17 metrics.h File Reference

Distance between stack trace threads. #include <stdbool.h>
Include dependency graph for metrics.h:



## **Data Structures**

• struct btp\_distances

A distance matrix of stack trace threads.

# **Typedefs**

- typedef int(\* btp\_gdb\_frame\_cmp\_type )(struct btp\_gdb\_frame \*, struct btp\_gdb\_frame \*)
- typedef float(\* btp\_dist\_thread\_type )(struct btp\_gdb\_thread \*, struct btp\_gdb\_thread \*)

### **Functions**

- float **btp\_gdb\_thread\_jarowinkler\_distance** (struct btp\_gdb\_thread \*thread1, struct btp\_gdb\_thread \*thread2)
- float **btp\_gdb\_thread\_jaccard\_distance** (struct btp\_gdb\_thread \*thread1, struct btp\_gdb\_thread \*thread2)
- int **btp\_gdb\_thread\_levenshtein\_distance** (struct btp\_gdb\_thread \*thread1, struct btp\_gdb\_thread \*thread2, bool transposition)
- float **btp\_gdb\_thread\_levenshtein\_distance\_f** (struct btp\_gdb\_thread \*thread1, struct btp\_gdb\_thread \*thread2)
- float **btp\_gdb\_thread\_jarowinkler\_distance\_custom** (struct btp\_gdb\_thread \*thread1, struct btp\_gdb\_thread \*thread2, btp\_gdb\_frame\_cmp\_type compare\_func)
- float **btp\_gdb\_thread\_jaccard\_distance\_custom** (struct btp\_gdb\_thread \*thread1, struct btp\_gdb\_thread \*thread2, btp\_gdb\_frame\_cmp\_type compare\_func)
- int **btp\_gdb\_thread\_levenshtein\_distance\_custom** (struct btp\_gdb\_thread \*thread1, struct btp\_gdb\_thread \*thread2, bool transposition, btp\_gdb\_frame\_cmp\_type compare\_func)
- struct btp\_distances \* btp\_distances\_new (int m, int n)
- struct btp\_distances \* btp\_distances\_dup (struct btp\_distances \*distances)
- void btp\_distances\_free (struct btp\_distances \*distances)
- float btp\_distances\_get\_distance (struct btp\_distances \*distances, int i, int j)
- void btp\_distances\_set\_distance (struct btp\_distances \*distances, int i, int j, float d)
- struct btp\_distances \* btp\_gdb\_threads\_compare (struct btp\_gdb\_thread \*\*threads, int m, int n, btp\_dist\_thread\_type dist\_func)

# 9.17.1 Detailed Description

Distance between stack trace threads.

# 9.17.2 Typedef Documentation

9.17.2.1 typedef float(\* btp\_dist\_thread\_type)(struct btp\_gdb\_thread \*, struct btp\_gdb\_thread \*)

A function which compares two threads.

## 9.17.3 Function Documentation

# 9.17.3.1 struct btp\_distances\* btp\_distances\_dup (struct btp\_distances \* distances) [read]

Creates a duplicate of the distances structure.

### **Parameters:**

distances It must be non-NULL pointer. The structure is not modified by calling this function.

### **Returns:**

This function never returns NULL.

## 9.17.3.2 void btp\_distances\_free (struct btp\_distances \* distances)

Releases the memory held by the distances structure.

### **Parameters:**

distances If the distances is NULL, no operation is performed.

# 9.17.3.3 float btp\_distances\_get\_distance (struct btp\_distances \* distances, int i, int j)

Gets the entry (i, j) from the distance matrix.

### **Parameters:**

distances It must be non-NULL pointer.

i Row in the matrix.

j Column in the matrix.

### **Returns:**

For entries (i, i) zero distance is returned and values returned for entries (i, j) and (j, i) are the same.

## 9.17.3.4 struct btp\_distances\* btp\_distances\_new (int *m*, int *n*) [read]

Creates and initializes a new distances structure.

### **Parameters:**

- m Number of rows.
- n Number of columns.

### **Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp\_distances\_free().

# 9.17.3.5 void btp\_distances\_set\_distance (struct btp\_distances \* distances, int i, int j, float d)

Sets the entry (i, j) from the distance matrix.

### **Parameters:**

distances It must be non-NULL pointer.

- *i* Row in the matrix.
- j Column in the matrix.
- d Distance.

# 9.17.3.6 struct btp\_distances\* btp\_gdb\_threads\_compare (struct btp\_gdb\_thread \*\* threads, int m, int n, btp\_dist\_thread\_type dist\_func) [read]

Creates a distances structure by comparing threads.

### **Parameters:**

threads Array of threads. They are not modified by calling this function.

- m Compare first m threads from the array with other threads.
- **n** Number of threads in the passed array.
- *dist\_func* Distance function which will be used to compare the threads. It's assumed to be symmetric and return zero distance for equal threads.

## **Returns:**

This function never returns NULL.

## 9.18 normalize.h File Reference

Normalization of stack traces.

## **Functions**

- void **btp normalize gdb thread** (struct btp gdb thread \*thread)
- void **btp\_normalize\_gdb\_stacktrace** (struct btp\_gdb\_stacktrace \*stacktrace)
- void btp\_normalize\_koops\_stacktrace (struct btp\_koops\_stacktrace \*stacktrace)
- struct btp\_gdb\_frame \* btp\_glibc\_thread\_find\_exit\_frame (struct btp\_gdb\_thread \*thread)
- void btp\_normalize\_gdb\_paired\_unknown\_function\_names (struct btp\_gdb\_thread \*thread1, struct btp\_gdb\_thread \*thread2)
- void btp\_gdb\_normalize\_optimize\_thread (struct btp\_gdb\_thread \*thread)

## 9.18.1 Detailed Description

Normalization of stack traces. Normalization changes stack traces with respect to similarity by removing unnecessary differences. Normalized stack traces can be used to compute clusters and similarity of stack traces.

#### 9.18.2 Function Documentation

#### 9.18.2.1 void btp\_gdb\_normalize\_optimize\_thread (struct btp\_gdb\_thread \* thread)

Remove frames which are not interesting in comparison with other threads.

## 9.18.2.2 struct btp\_gdb\_frame\* btp\_glibc\_thread\_find\_exit\_frame (struct btp\_gdb\_thread \* thread) [read]

Checks whether the thread it contains some function used to exit application. If a frame with the function is found, it is returned. If there are multiple frames with abort function, the lowest one is returned.

#### **Returns:**

Returns NULL if such a frame is not found.

## 9.18.2.3 void btp\_normalize\_gdb\_paired\_unknown\_function\_names (struct btp\_gdb\_thread \* thread1, struct btp\_gdb\_thread \* thread2)

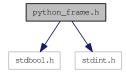
Renames unknown function names ("??") that are between the same function names to be treated as similar in later comparison. Leaves unpair unknown functions unchanged.

## 9.19 python\_frame.h File Reference

Python frame structure and related algorithms. #include <stdbool.h>

#include <stdint.h>

Include dependency graph for python\_frame.h:



#### **Data Structures**

• struct btp\_python\_frame

#### **Functions**

- struct btp\_python\_frame \* btp\_python\_frame\_new ()
- void btp\_python\_frame\_init (struct btp\_python\_frame \*frame)
- void btp\_python\_frame\_free (struct btp\_python\_frame \*frame)
- struct btp\_python\_frame \* btp\_python\_frame\_dup (struct btp\_python\_frame \*frame, bool siblings)

## 9.19.1 Detailed Description

Python frame structure and related algorithms.

#### 9.19.2 Function Documentation

## 9.19.2.1 struct btp\_python\_frame\* btp\_python\_frame\_dup (struct btp\_python\_frame \* frame, bool siblings) [read]

Creates a duplicate of the frame.

#### **Parameters:**

frame It must be non-NULL pointer. The frame is not modified by calling this function.

*siblings* Whether to duplicate also siblings referenced by frame->next. If false, frame->next is not duplicated for the new frame, but it is set to NULL.

#### **Returns:**

This function never returns NULL. The returned duplicate frame must be released by calling the function btp\_python\_frame\_free().

#### 9.19.2.2 void btp\_python\_frame\_free (struct btp\_python\_frame \* frame)

Releases the memory held by the frame. The frame siblings are not released.

#### **Parameters:**

frame If the frame is NULL, no operation is performed.

## 9.19.2.3 void btp\_python\_frame\_init (struct btp\_python\_frame \* frame)

Initializes all members of the frame structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a frame structure placed on the stack.

## 9.19.2.4 struct btp\_python\_frame\* btp\_python\_frame\_new() [read]

Creates and initializes a new frame structure.

#### **Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp\_python\_frame\_free().

## 9.20 python\_stacktrace.h File Reference

Python stack trace structure and related algorithms. #include <stdint.h>
Include dependency graph for python\_stacktrace.h:



#### **Data Structures**

• struct btp\_python\_stacktrace

### **Functions**

- struct btp\_python\_stacktrace \* btp\_python\_stacktrace\_new ()
- void btp\_python\_stacktrace\_init (struct btp\_python\_stacktrace \*stacktrace)
- void btp\_python\_stacktrace\_free (struct btp\_python\_stacktrace \*stacktrace)
- struct btp\_python\_stacktrace \* btp\_python\_stacktrace\_dup (struct btp\_python\_stacktrace \*stacktrace)

## 9.20.1 Detailed Description

Python stack trace structure and related algorithms.

#### 9.20.2 Function Documentation

## 9.20.2.1 struct btp\_python\_stacktrace\* btp\_python\_stacktrace\_dup (struct btp\_python\_stacktrace \* stacktrace) [read]

Creates a duplicate of the stacktrace.

### **Parameters:**

stacktrace The stacktrace to be copied. It's not modified by this function.

#### **Returns:**

This function never returns NULL. The returned duplicate must be released by calling the function btp\_python\_stacktrace\_free().

## 9.20.2.2 void btp\_python\_stacktrace\_free (struct btp\_python\_stacktrace \* stacktrace)

Releases the memory held by the stacktrace and its frames.

#### **Parameters:**

stacktrace If the stacktrace is NULL, no operation is performed.

## 9.20.2.3 void btp\_python\_stacktrace\_init (struct btp\_python\_stacktrace \* stacktrace)

Initializes all members of the stacktrace structure to their default values. No memory is released, members are simply overwritten. This is useful for initializing a stacktrace structure placed on the stack.

## $9.20.2.4 \quad struct \ btp\_python\_stacktrace* \ btp\_python\_stacktrace\_new \ () \quad \hbox{\tt [read]}$

Creates and initializes a new stacktrace structure.

#### **Returns:**

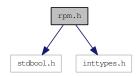
It never returns NULL. The returned pointer must be released by calling the function btp\_python\_stacktrace\_free().

## 9.21 rpm.h File Reference

RPM-related structures and utilities. #include <stdbool.h>

#include <inttypes.h>

Include dependency graph for rpm.h:



#### **Data Structures**

- struct btp\_rpm\_consistency
- struct btp\_rpm\_package

#### **Functions**

- struct btp\_rpm\_package \* btp\_rpm\_package\_new ()
- void btp\_rpm\_package\_init (struct btp\_rpm\_package \*package)
- void btp\_rpm\_package\_free (struct btp\_rpm\_package \*package, bool recursive)
- struct btp\_rpm\_package \* btp\_rpm\_package\_append (struct btp\_rpm\_package \*dest, struct btp\_rpm\_package \*item)
- struct btp\_rpm\_package \* btp\_rpm\_package\_get\_by\_name (const char \*name, char \*\*error\_-message)
- struct btp\_rpm\_package \* btp\_rpm\_package\_get\_by\_path (const char \*path, char \*\*error\_message)
- struct btp\_rpm\_consistency \* btp\_rpm\_consistency\_new ()
- void **btp\_rpm\_consistency\_init** (struct btp\_rpm\_consistency \*consistency)
- void btp\_rpm\_consistency\_free (struct btp\_rpm\_consistency \*consistency, bool recursive)

### 9.21.1 Detailed Description

RPM-related structures and utilities.

### 9.21.2 Function Documentation

9.21.2.1 struct btp\_rpm\_package\* btp\_rpm\_package\_append (struct btp\_rpm\_package \* dest, struct btp\_rpm\_package \* item) [read]

Appends 'item' at the end of the list 'dest'.

#### **Returns:**

This function returns the 'dest' package. If 'dest' is NULL, it returns the 'item' package.

9.22 sha1.h File Reference

## 9.22 sha1.h File Reference

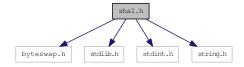
An implementation of SHA-1 cryptographic hash function. #include <byteswap.h>

#include <stdlib.h>

#include <stdint.h>

#include <string.h>

Include dependency graph for sha1.h:



#### **Data Structures**

• struct btp\_sha1\_state

Internal state of a SHA-1 hash algorithm.

## **Defines**

- #define BTP\_SHA1\_RESULT\_BIN\_LEN (5 \* 4)
- #define **BTP\_SHA1\_RESULT\_LEN** (5 \* 4 \* 2 + 1)

## **Functions**

- void **btp\_sha1\_begin** (struct btp\_sha1\_state \*state)
- void btp\_sha1\_hash (struct btp\_sha1\_state \*state, const void \*buffer, size\_t len)
- void btp\_sha1\_end (struct btp\_sha1\_state \*state, void \*resbuf)

## 9.22.1 Detailed Description

An implementation of SHA-1 cryptographic hash function.

## 9.23 strbuf.h File Reference

A string buffer structure and related algorithms. #include <stdarg.h> Include dependency graph for strbuf.h:



#### **Data Structures**

• struct btp\_strbuf

A resizable string buffer.

#### **Functions**

- struct btp\_strbuf \* btp\_strbuf\_new ()
- void btp\_strbuf\_init (struct btp\_strbuf \*strbuf)
- void btp\_strbuf\_free (struct btp\_strbuf \*strbuf)
- char \* btp\_strbuf\_free\_nobuf (struct btp\_strbuf \*strbuf)
- void btp\_strbuf\_clear (struct btp\_strbuf \*strbuf)
- void btp\_strbuf\_grow (struct btp\_strbuf \*strbuf, int num)
- struct btp\_strbuf \* btp\_strbuf\_append\_char (struct btp\_strbuf \*strbuf, char c)
- struct btp\_strbuf \* btp\_strbuf\_append\_str (struct btp\_strbuf \*strbuf, const char \*str)
- struct btp\_strbuf \* btp\_strbuf\_prepend\_str (struct btp\_strbuf \*strbuf, const char \*str)
- struct btp\_strbuf \* btp\_strbuf\_append\_strf (struct btp\_strbuf \*strbuf, const char \*format,...)
- struct btp\_strbuf \* btp\_strbuf\_append\_strfv (struct btp\_strbuf \*strbuf, const char \*format, va\_list p)
- struct btp\_strbuf \* btp\_strbuf\_prepend\_strf (struct btp\_strbuf \*strbuf, const char \*format,...)
- struct btp\_strbuf \* btp\_strbuf\_prepend\_strfv (struct btp\_strbuf \*strbuf, const char \*format, va\_list p)

#### 9.23.1 Detailed Description

A string buffer structure and related algorithms.

#### 9.23.2 Function Documentation

## 9.23.2.1 struct btp\_strbuf\* btp\_strbuf\_append\_char (struct btp\_strbuf\* strbuf\*, char c) [read]

The current content of the string buffer is extended by adding a character c at its end.

## 9.23.2.2 struct btp\_strbuf\* btp\_strbuf\_append\_str (struct btp\_strbuf \* strbuf, const char \* str) [read]

The current content of the string buffer is extended by adding a string str at its end.

## 9.23.2.3 struct btp\_strbuf\* btp\_strbuf\_append\_strf (struct btp\_strbuf\* strbuf\*, const char \* format, ...) [read]

The current content of the string buffer is extended by adding a sequence of data formatted as the format argument specifies.

## 9.23.2.4 struct btp\_strbuf\* btp\_strbuf\_append\_strfv (struct btp\_strbuf \* strbuf, const char \* format, va\_list p) [read]

Same as btp\_strbuf\_append\_strf except that va\_list is used instead of variable number of arguments.

### 9.23.2.5 void btp\_strbuf\_clear (struct btp\_strbuf \* strbuf)

The string content is set to an empty string, erasing any previous content and leaving its length at 0 characters.

### 9.23.2.6 void btp\_strbuf\_free (struct btp\_strbuf \* strbuf)

Releases the memory held by the string buffer.

#### **Parameters:**

strbuf If the strbuf is NULL, no operation is performed.

#### 9.23.2.7 char\* btp\_strbuf\_free\_nobuf (struct btp\_strbuf \* strbuf)

Releases the strbuf, but not the internal buffer. The internal string buffer is returned. Caller is responsible to release the returned memory using free().

#### 9.23.2.8 void btp\_strbuf\_grow (struct btp\_strbuf \* strbuf, int num)

Ensures that the buffer can be extended by num characters without dealing with malloc/realloc.

#### 9.23.2.9 void btp\_strbuf\_init (struct btp\_strbuf \* strbuf)

Initializes all members of the strbuf structure to their default values. No memory is released, members are simply overritten. This is useful for initializing a strbuf structure placed on the stack.

#### 9.23.2.10 struct btp\_strbuf\* btp\_strbuf\_new() [read]

Creates and initializes a new string buffer.

#### **Returns:**

It never returns NULL. The returned pointer must be released by calling the function btp\_strbuf\_free().

## 9.23.2.11 struct btp\_strbuf\* btp\_strbuf\_prepend\_str (struct btp\_strbuf \* strbuf, const char \* str) [read]

The current content of the string buffer is extended by inserting a string str at its beginning.

## 9.23.2.12 struct btp\_strbuf\* btp\_strbuf\_prepend\_strf (struct btp\_strbuf\* strbuf\*, const char \* format\*, ...) [read]

The current content of the string buffer is extended by inserting a sequence of data formatted as the format argument specifies at the buffer beginning.

## 9.23.2.13 struct btp\_strbuf\* btp\_strbuf\_prepend\_strfv (struct btp\_strbuf \* strbuf, const char \* format, va\_list p) [read]

Same as btp\_strbuf\_prepend\_strf except that va\_list is used instead of variable number of arguments.

## 9.24 unstrip.h File Reference

Parser for the output of the unstrip utility. #include <inttypes.h> Include dependency graph for unstrip.h:



## **Data Structures**

• struct btp\_unstrip\_entry

Core dump memory layout as reported by the unstrip utility.

## **Functions**

- struct btp\_unstrip\_entry \* btp\_unstrip\_parse (const char \*unstrip\_output)
- struct btp\_unstrip\_entry \* **btp\_unstrip\_find\_address** (struct btp\_unstrip\_entry \*entries, uint64\_t address)
- void **btp\_unstrip\_free** (struct btp\_unstrip\_entry \*entries)

## 9.24.1 Detailed Description

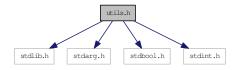
Parser for the output of the unstrip utility.

## 9.25 utils.h File Reference

Various utility functions, macros and variables that do not fit elsewhere. #include <stdlib.h>

#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>

Include dependency graph for utils.h:



#### **Defines**

- #define **BTP\_lower** "abcdefghijklmnopqrstuvwxyz"
- #define **BTP upper** "ABCDEFGHIJKLMNOPORSTUVWXYZ"
- #define BTP\_alpha BTP\_lower BTP\_upper
- #define **BTP\_space** "  $\t \r \n \v \f$ "
- #define **BTP\_digit** "0123456789"
- #define BTP\_alnum BTP\_alpha BTP\_digit

#### **Functions**

- void \* btp\_malloc (size\_t size)
- void \* btp\_mallocz (size\_t size)
- void \* btp\_realloc (void \*ptr, size\_t size)
- char \* btp\_vasprintf (const char \*format, va\_list p)
- char \* btp\_asprintf (const char \*format,...)
- char \* btp\_strdup (const char \*s)
- char \* btp\_strndup (const char \*s, size\_t n)
- void **btp\_struniq** (char \*\*strings, size\_t \*size)
- int btp\_strcmp0 (const char \*s1, const char \*s2)
- char \* btp\_strchr\_location (const char \*s, int c, int \*line, int \*column)
- char \* btp\_strstr\_location (const char \*haystack, const char \*needle, int \*line, int \*column)
- size\_t btp\_strspn\_location (const char \*s, const char \*accept, int \*line, int \*column)
- char \* btp\_file\_to\_string (const char \*filename)
- bool btp\_skip\_char (const char \*\*input, char c)
- bool btp\_skip\_char\_limited (const char \*\*input, const char \*allowed)
- bool btp\_parse\_char\_limited (const char \*\*input, const char \*allowed, char \*result)
- int btp\_skip\_char\_sequence (const char \*\*input, char c)
- int btp\_skip\_char\_span (const char \*\*input, const char \*chars)
- int btp\_skip\_char\_span\_location (const char \*\*input, const char \*chars, int \*line, int \*column)
- int btp\_parse\_char\_span (const char \*\*input, const char \*accept, char \*\*result)
- int btp\_skip\_char\_cspan (const char \*\*input, const char \*reject)
- bool btp\_parse\_char\_cspan (const char \*\*input, const char \*reject, char \*\*result)
- int btp\_skip\_string (const char \*\*input, const char \*string)

9.25 utils.h File Reference

- bool btp\_parse\_string (const char \*\*input, const char \*string, char \*\*result)
- char btp\_parse\_digit (const char \*\*input)
- int btp\_skip\_uint (const char \*\*input)
- int btp\_parse\_uint32 (const char \*\*input, uint32\_t \*result)
- int **btp\_parse\_uint64** (const char \*\*input, uint64\_t \*result)
- int btp\_skip\_hexadecimal\_uint (const char \*\*input)
- int btp\_skip\_hexadecimal\_0xuint (const char \*\*input)
- int btp\_parse\_hexadecimal\_uint64 (const char \*\*input, uint64\_t \*result)
- int btp\_parse\_hexadecimal\_0xuint64 (const char \*\*input, uint64\_t \*result)
- char \* btp\_skip\_whitespace (const char \*s)
- char \* btp\_skip\_non\_whitespace (const char \*s)
- char \* btp\_bin2hex (char \*dst, const char \*str, int count)
- char \* btp\_indent (const char \*input, int spaces)
- char \* btp\_indent\_except\_first\_line (const char \*input, int spaces)

#### **Variables**

• bool btp\_debug\_parser

### 9.25.1 Detailed Description

Various utility functions, macros and variables that do not fit elsewhere.

#### 9.25.2 Function Documentation

## 9.25.2.1 char\* btp\_asprintf (const char \* format, ...)

Never returns NULL.

#### 9.25.2.2 char\* btp\_bin2hex (char \* dst, const char \* str, int count)

Emit a string of hex representation of bytes.

#### 9.25.2.3 char\* btp\_file\_to\_string (const char \* filename)

Loads file contents to a string.

#### **Returns:**

File contents. If file opening/reading fails, NULL is returned.

#### 9.25.2.4 void\* btp\_malloc (size\_t size)

Never returns NULL.

#### 9.25.2.5 void\* btp mallocz (size t size)

Never returns NULL.

#### 9.25.2.6 bool btp\_parse\_char\_cspan (const char \*\* input, const char \* reject, char \*\* result)

If the input contains characters which are not in string reject, create a string from this sequence and store it to the result, move the input pointer after the sequence, and return true. Otherwise do not modify the input and return false.

If this function returns true, the caller is responsible to free the result.

### 9.25.2.7 bool btp\_parse\_char\_limited (const char \*\* input, const char \* allowed, char \* result)

If the input contains one of allowed characters, store the character to the result, move the input pointer after that character, and return true. Otherwise do not modify the input and return false.

#### 9.25.2.8 int btp\_parse\_char\_span (const char \*\* input, const char \* accept, char \*\* result)

If the input contains one or more characters from string accept, create a string from this sequence and store it to the result, move the input pointer after the sequence, and return the length of the sequence. Otherwise do not modify the input and return 0.

If this function returns nonzero value, the caller is responsible to free the result.

## 9.25.2.9 char btp\_parse\_digit (const char \*\* input)

If the input contains digit 0-9, return it as a character and move the input pointer after it. Otherwise return " and do not modify the input.

#### 9.25.2.10 int btp\_parse\_hexadecimal\_0xuint64 (const char \*\* input, uint64\_t \* result)

If the input contains 0x[0-9a-f]+, parse the number, and move the input pointer after it. Otherwise do not modify the input.

#### **Returns:**

The number of characters read from input. 0 if the input does not contain a hexadecimal number.

## 9.25.2.11 int btp\_parse\_hexadecimal\_uint64 (const char \*\* input, uint64\_t \* result)

If the input contains [0-9a-f]+, parse the number, and move the input pointer after it. Otherwise do not modify the input.

#### **Returns:**

The number of characters read from input. 0 if the input does not contain a hexadecimal number.

#### 9.25.2.12 bool btp parse string (const char \*\* input, const char \* string, char \*\* result)

If the input contains the string, copy the string to result, move the input pointer after the string, and return true. Otherwise do not modify the input and return false.

If this function returns true, the caller is responsible to free the result.

9.25 utils.h File Reference 123

#### 9.25.2.13 int btp\_parse\_uint32 (const char \*\* input, uint32\_t \* result)

If the input contains [0-9]+, parse it, move the input pointer after the number.

#### Returns:

Number of parsed characters. 0 if input does not contain a number.

#### 9.25.2.14 void\* btp\_realloc (void \* ptr, size\_t size)

Never returns NULL.

### 9.25.2.15 bool btp\_skip\_char (const char \*\* input, char c)

If the input contains character c in the current positon, move the input pointer after the character, and return true. Otherwise do not modify the input and return false.

#### 9.25.2.16 int btp\_skip\_char\_cspan (const char \*\* input, const char \* reject)

If the input contains one or more characters which are not present in string reject, move the input pointer after the sequence. Otherwise do not modify the input.

#### **Returns:**

The number of characters skipped.

#### 9.25.2.17 bool btp\_skip\_char\_limited (const char \*\* input, const char \* allowed)

If the input contains one of allowed characters, move the input pointer after that character, and return true. Otherwise do not modify the input and return false.

#### 9.25.2.18 int btp\_skip\_char\_sequence (const char \*\* input, char c)

If the input contains the character c one or more times, update it so that the characters are skipped. Returns the number of characters skipped, thus zero if \*\*input does not contain c.

#### 9.25.2.19 int btp\_skip\_char\_span (const char \*\* input, const char \* chars)

If the input contains one or more characters from string chars, move the input pointer after the sequence. Otherwise do not modify the input.

#### **Returns:**

The number of characters skipped.

## 9.25.2.20 int btp\_skip\_char\_span\_location (const char \*\* input, const char \* chars, int \* line, int \* column)

If the input contains one or more characters from string chars, move the input pointer after the sequence. Otherwise do not modify the input.

#### **Parameters:**

line Starts from 1. Corresponds to the returned number.column Starts from 0. Corresponds to the returned number.

#### **Returns:**

The number of characters skipped.

#### 9.25.2.21 int btp\_skip\_hexadecimal\_0xuint (const char \*\* input)

If the input contains 0x[0-9a-f]+, move the input pointer after that.

#### **Returns:**

The number of characters processed from input. 0 if the input does not contain a hexadecimal number.

#### 9.25.2.22 int btp\_skip\_hexadecimal\_uint (const char \*\* input)

If the input contains [0-9a-f]+, move the input pointer after that.

#### **Returns:**

The number of characters processed from input. 0 if the input does not contain a hexadecimal number.

## 9.25.2.23 int btp\_skip\_string (const char \*\* input, const char \* string)

If the input contains the string, move the input pointer after the sequence. Otherwise do not modify the input.

#### **Returns:**

Number of characters skipped. 0 if the input does not contain the string.

#### 9.25.2.24 int btp skip uint (const char \*\* input)

If the input contains [0-9]+, move the input pointer after the number.

#### **Returns:**

The number of skipped characters. 0 if input does not start with a digit.

9.25 utils.h File Reference 125

#### 9.25.2.25 char\* btp\_strchr\_location (const char \* s, int c, int \* line, int \* column)

A strchr() variant providing line and column in the string s indicating where the char c was found.

#### **Parameters:**

*line* Starts from 1. Its value is valid only when this function does not return NULL. *column* Starts from 0. Its value is valid only when this function does not return NULL.

## 9.25.2.26 int btp\_strcmp0 (const char \*s1, const char \*s2)

A strcmp() variant that works also with NULL parameters. NULL is considered to be less than a string.

#### 9.25.2.27 char\* btp\_strdup (const char \* s)

Never returns NULL.

#### 9.25.2.28 char\* btp\_strndup (const char \* s, size\_t n)

Never returns NULL.

#### 9.25.2.29 size\_t btp\_strspn\_location (const char \* s, const char \* accept, int \* line, int \* column)

A strspn() variant providing line and column of the string s which corresponds to the returned length.

## **Parameters:**

*line* Starts from 1. *column* Starts from 0.

## 9.25.2.30 char\* btp\_strstr\_location (const char \* haystack, const char \* needle, int \* line, int \* column)

A strstr() variant providing line and column of the haystick indicating where the needle was found.

#### **Parameters:**

*line* Starts from 1. Its value is valid only when this function does not return NULL. *column* Starts from 0. Its value is valid only when this function does not return NULL.

### 9.25.2.31 char\* btp\_vasprintf (const char \* format, va\_list p)

Never returns NULL.

#### **9.25.3** Variable Documentation

## 9.25.3.1 bool btp\_debug\_parser

Debugging output to stdout while parsing. Default value is false.

## **Chapter 10**

# **Example Documentation**

## 10.1 /home/karel/devel/btparser/lib/koops\_frame.h

Timestamp may be present in the oops lines. [123456.654321] [ 65.470000]

# Chapter 11

# **Known Bugs**

Empty.

130 Known Bugs

# **Chapter 12**

# Wishlist

Stack trace for kerneloopses, Python, and Java.

## **Index**

address	btp_core_stacktrace_free
btp_core_frame, 30	core_stacktrace.h, 69
btp_elf_plt_entry, 39	btp_core_stacktrace_get_thread_count
btp_gdb_frame, 40	core_stacktrace.h, 70
btp_koops_frame, 47	btp_core_stacktrace_init
alloc	core_stacktrace.h, 70
btp_strbuf, 59	btp_core_stacktrace_new
T — · · · · · · · · · · · · · · · · · ·	core_stacktrace.h, 70
btp_asprintf	btp_core_stacktrace_parse
utils.h, 121	core_stacktrace.h, 70
btp_bin2hex	btp_core_stacktrace_to_json
utils.h, 121	core_stacktrace.h, 70
btp_callgraph, 27	btp_core_thread, 34
callees, 27	frames, 34
btp_callgraph_extend	next, 34
callgraph.h, 62	btp_core_thread_append
btp_cluster, 29	core_thread.h, 71
btp_cluster_free	btp_core_thread_cmp
cluster.h, 63	core_thread.h, 71
btp_cluster_new	btp_core_thread_dup
cluster.h, 63	core_thread.h, 72
btp_core_frame, 30	btp_core_thread_free
address, 30	core_thread.h, 72
build_id, 30	btp_core_thread_get_frame_count
fingerprint, 30	core_thread.h, 72
next, 30	btp_core_thread_init
btp_core_frame_append	core_thread.h, 72
core_frame.h, 66	btp_core_thread_new
btp_core_frame_cmp	core_thread.h, 72
core_frame.h, 66	btp_deb_package, 35
btp_core_frame_dup	btp_debug_parser
core_frame.h, 67	utils.h, 125
btp_core_frame_free	btp_dendrogram, 36
core_frame.h, 67	merge_levels, 36
btp_core_frame_init	btp_dendrogram_cut
core_frame.h, 67	cluster.h, 63
btp_core_frame_new	btp_dendrogram_free
core_frame.h, 67	cluster.h, 64
btp_core_frame_to_json	btp_dendrogram_new
core_frame.h, 67	cluster.h, 64
btp_core_stacktrace, 32	btp_disasm_get_function_instructions
crash_thread, 32	disasm.h, 74
signal, 32	btp_dist_thread_type
btp_core_stacktrace_dup	metrics.h, 107
core_stacktrace.h, 69	btp_distances, 37
Core_Stacktrace.ii, 09	orp_distances, 57

btp_distances_cluster_objects	btp_gdb_frame_parse_address_in_function
cluster.h, 64	gdb_frame.h, 80
btp_distances_dup	btp_gdb_frame_parse_file_location
metrics.h, 107	gdb_frame.h, 80
btp_distances_free	btp_gdb_frame_parse_frame_start
metrics.h, 107	gdb_frame.h, 81
btp_distances_get_distance	btp_gdb_frame_parse_function_call
metrics.h, 107	gdb_frame.h, 81
btp_distances_new	btp_gdb_frame_parse_function_name
metrics.h, 107	gdb_frame.h, 81
btp_distances_set_distance	btp_gdb_frame_parse_function_name_braces
metrics.h, 108	gdb_frame.h, 82
btp_elf_fde, 38	btp_gdb_frame_parse_function_name_chunk
length, 38	gdb_frame.h, 82
start_address, 38	btp_gdb_frame_parse_function_name_template
btp_elf_get_eh_frame	gdb_frame.h, 82
elves.h, 75	btp_gdb_frame_parse_header
btp_elf_get_procedure_linkage_table	gdb_frame.h, 82
elves.h, 76	btp_gdb_frame_parseadd_operator
btp_elf_plt_entry, 39 address, 39	gdb_frame.h, 83
	btp_gdb_frame_remove_func_prefix
symbol_name, 39	gdb_frame.h, 83
btp_file_to_string	btp_gdb_frame_skip_function_args
utils.h, 121	gdb_frame.h, 83
btp_gdb_frame, 40	btp_gdb_normalize_optimize_thread
address, 40	normalize.h, 109
function_name, 40	btp_gdb_sharedlib, 42
function_type, 40	btp_gdb_sharedlib_append
library_name, 40	gdb_sharedlib.h, 84
next, 40	btp_gdb_sharedlib_count
number, 41	gdb_sharedlib.h, 85
signal_handler_called, 41	btp_gdb_sharedlib_dup
source_file, 41	gdb_sharedlib.h, 85
source_line, 41	btp_gdb_sharedlib_find_address
btp_gdb_frame_append	gdb_sharedlib.h, 85
gdb_frame.h, 78	btp_gdb_sharedlib_free
btp_gdb_frame_append_to_str	gdb_sharedlib.h, 85
gdb_frame.h, 78	btp_gdb_sharedlib_init
btp_gdb_frame_calls_func	gdb_sharedlib.h, 85
gdb_frame.h, 78	btp_gdb_sharedlib_new
btp_gdb_frame_cmp	gdb_sharedlib.h, 85
gdb_frame.h, 78	btp_gdb_sharedlib_parse
btp_gdb_frame_cmp_simple	gdb_sharedlib.h, 85
gdb_frame.h, 79	btp_gdb_stacktrace, 43
btp_gdb_frame_dup	crash, 43
gdb_frame.h, 79	libs, 43
btp_gdb_frame_free	btp_gdb_stacktrace_dup
gdb_frame.h, 79	gdb_stacktrace.h, 88
btp_gdb_frame_init	btp_gdb_stacktrace_find_crash_thread
gdb_frame.h, 79	gdb_stacktrace.h, 88
btp_gdb_frame_new	btp_gdb_stacktrace_free
gdb_frame.h, 80	gdb_stacktrace.h, 88
btp_gdb_frame_parse	btp_gdb_stacktrace_get_crash_frame
gdb_frame.h, 80	gdb_stacktrace.h, 88
5	500_500Kirucc.ii, 00

btp_gdb_stacktrace_get_duplication_hash	gdb_thread.h, 96
gdb_stacktrace.h, 88	btp_gdb_thread_remove_frame
btp_gdb_stacktrace_get_optimized_thread	gdb_thread.h, 96
gdb_stacktrace.h, 89	btp_gdb_thread_remove_frames_above
btp_gdb_stacktrace_get_thread_count	gdb_thread.h, 96
gdb_stacktrace.h, 89	btp_gdb_thread_remove_frames_below_n
btp_gdb_stacktrace_init	gdb_thread.h, 96
gdb_stacktrace.h, 89	btp_gdb_thread_skip_lwp
btp_gdb_stacktrace_limit_frame_depth	gdb_thread.h, 96
gdb_stacktrace.h, 89	btp_gdb_threads_compare
btp_gdb_stacktrace_new	metrics.h, 108
gdb_stacktrace.h, 89	btp_glibc_thread_find_exit_frame
btp_gdb_stacktrace_parse	normalize.h, 109
gdb_stacktrace.h, 90	btp_json_settings, 45
btp_gdb_stacktrace_parse_header	btp_json_value, 46
gdb_stacktrace.h, 90	btp_koops_frame, 47
btp_gdb_stacktrace_quality_complex	address, 47
gdb_stacktrace.h, 91	from_address, 47
btp_gdb_stacktrace_quality_simple	from_function_name, 47
gdb_stacktrace.h, 91	
	from_module_name, 47
btp_gdb_stacktrace_remove_threads_except_one	function_name, 47
gdb_stacktrace.h, 91	module_name, 48
btp_gdb_stacktrace_set_libnames	reliable, 48
gdb_stacktrace.h, 91	btp_koops_frame_append
btp_gdb_stacktrace_to_text	koops_frame.h, 98
gdb_stacktrace.h, 92	btp_koops_frame_cmp
btp_gdb_thread, 44	koops_frame.h, 98
frames, 44	btp_koops_frame_dup
next, 44	koops_frame.h, 99
btp_gdb_thread_append	btp_koops_frame_free
gdb_thread.h, 94	koops_frame.h, 99
btp_gdb_thread_append_to_str	btp_koops_frame_init
gdb_thread.h, 94	koops_frame.h, 99
btp_gdb_thread_cmp	btp_koops_frame_new
gdb_thread.h, 94	koops_frame.h, 99
btp_gdb_thread_dup	btp_koops_frame_to_json
gdb_thread.h, 94	koops_frame.h, 99
btp_gdb_thread_format_funs	btp_koops_stacktrace, 49
gdb_thread.h, 94	modules, 49
btp_gdb_thread_free	taint_mce, 49
gdb_thread.h, 94	taint_module_proprietary, 50
btp_gdb_thread_get_frame_count	taint_page_release, 50
gdb_thread.h, 95	btp_koops_stacktrace_dup
btp_gdb_thread_init	koops_stacktrace.h, 101
gdb_thread.h, 95	btp_koops_stacktrace_free
btp_gdb_thread_new	koops_stacktrace.h, 101
gdb_thread.h, 95	btp_koops_stacktrace_get_frame_count
btp_gdb_thread_parse	koops stacktrace.h, 102
gdb_thread.h, 95	btp_koops_stacktrace_init
btp_gdb_thread_parse_funs	koops_stacktrace.h, 102
gdb_thread.h, 95	btp_koops_stacktrace_new
btp_gdb_thread_quality	koops_stacktrace_hew
gdb_thread.h, 95	btp_koops_stacktrace_parse
•	
btp_gdb_thread_quality_counts	koops_stacktrace.h, 102

btp_koops_stacktrace_remove_frame	btp_python_stacktrace_dup
koops_stacktrace.h, 102	python_stacktrace.h, 112
btp_location, 51	btp_python_stacktrace_free
column, 51	python_stacktrace.h, 112
line, 51	btp_python_stacktrace_init
message, 51	python_stacktrace.h, 112
btp_location_add	btp_python_stacktrace_new
location.h, 103	python_stacktrace.h, 113
btp_location_add_ext	btp_realloc
location.h, 104	utils.h, 123
btp_location_cmp	btp_report, 55
location.h, 104	btp_rpm_consistency, 56
btp_location_eat_char	btp_rpm_package, 57
location.h, 104	btp_rpm_package_append
btp_location_eat_char_ext	rpm.h, 114
location.h, 104	btp_sha1_state, 58
btp_location_init	btp_skip_char
location.h, 104	utils.h, 123
btp_location_to_string	btp_skip_char_cspan
location.h, 105	utils.h, 123
btp_malloc	btp_skip_char_limited
utils.h, 121	utils.h, 123
btp_mallocz	btp_skip_char_sequence
utils.h, 121	utils.h, 123
btp_normalize_gdb_paired_unknown_function	btp_skip_char_span
names	utils.h, 123
normalize.h, 109	btp_skip_char_span_location
btp_operating_system, 52	utils.h, 123
btp_parse_char_cspan	btp_skip_hexadecimal_0xuint
utils.h, 121	utils.h, 124
btp_parse_char_limited	btp_skip_hexadecimal_uint
utils.h, 122	utils.h, 124
btp_parse_char_span	btp_skip_string
utils.h, 122 btp_parse_digit	utils.h, 124 btp_skip_uint
utils.h, 122	utils.h, 124
btp_parse_hexadecimal_0xuint64	btp_strbuf, 59
utils.h, 122	alloc, 59
btp_parse_hexadecimal_uint64	len, 59
utils.h, 122	btp_strbuf_append_char
btp_parse_string	strbuf.h, 116
utils.h, 122	btp_strbuf_append_str
btp_parse_uint32	strbuf.h, 116
utils.h, 122	btp_strbuf_append_strf
btp_python_frame, 53	strbuf.h, 116
btp_python_frame_dup	btp_strbuf_append_strfv
python_frame.h, 110	strbuf.h, 117
btp_python_frame_free	btp_strbuf_clear
python_frame.h, 110	strbuf.h, 117
btp_python_frame_init	btp_strbuf_free
python_frame.h, 111	strbuf.h, 117
btp_python_frame_new	btp_strbuf_free_nobuf
python_frame.h, 111	strbuf.h, 117
btp_python_stacktrace, 54	btp_strbuf_grow
· ·	. – —

strbuf.h, 117	btp_core_stacktrace_get_thread_count, 70
btp_strbuf_init	btp_core_stacktrace_init, 70
strbuf.h, 117	btp_core_stacktrace_new, 70
btp_strbuf_new	btp_core_stacktrace_parse, 70
strbuf.h, 117	btp_core_stacktrace_to_json, 70
btp_strbuf_prepend_str	core_thread.h, 71
strbuf.h, 117	btp_core_thread_append, 71
btp_strbuf_prepend_strf	btp_core_thread_cmp, 71
strbuf.h, 118	btp_core_thread_dup, 72
btp_strbuf_prepend_strfv	btp_core_thread_free, 72
strbuf.h, 118	btp_core_thread_get_frame_count, 72
btp_strchr_location	btp_core_thread_init, 72
utils.h, 124	btp_core_thread_new, 72
btp_strcmp0	crash
utils.h, 125	btp_gdb_stacktrace, 43
btp_strdup	crash_thread
utils.h, 125	btp_core_stacktrace, 32
btp_strndup	огр_соге_змектисе, 32
utils.h, 125	deb.h, 73
btp_strspn_location	disasm.h, 74
utils.h, 125	btp_disasm_get_function_instructions, 74
btp_strstr_location	otp_disasin_get_function_msu detions, 74
utils.h, 125	elves.h, 75
btp_unstrip_entry, 60	btp_elf_get_eh_frame, 75
	btp_elf_get_procedure_linkage_table, 76
btp_vasprintf	orp_en_ger_procedure_mikage_table, 70
utils.h, 125	fingerprint
build_id	btp_core_frame, 30
btp_core_frame, 30	frames
asllaas	
callees	btp_core_thread, 34
btp_callgraph, 27	btp_gdb_thread, 44
callgraph.h, 61	from_address
btp_callgraph_extend, 62	btp_koops_frame, 47
cluster.h, 63	from_function_name
btp_cluster_free, 63	btp_koops_frame, 47
btp_cluster_new, 63	from_module_name
btp_dendrogram_cut, 63	btp_koops_frame, 47
btp_dendrogram_free, 64	function_name
btp_dendrogram_new, 64	btp_gdb_frame, 40
btp_distances_cluster_objects, 64	btp_koops_frame, 47
column	function_type
btp_location, 51	btp_gdb_frame, 40
core_fingerprint.h, 65	
core_frame.h, 66	gdb_frame.h, 77
btp_core_frame_append, 66	btp_gdb_frame_append, 78
btp_core_frame_cmp, 66	btp_gdb_frame_append_to_str, 78
btp_core_frame_dup, 67	btp_gdb_frame_calls_func, 78
btp_core_frame_free, 67	btp_gdb_frame_cmp, 78
btp_core_frame_init, 67	btp_gdb_frame_cmp_simple, 79
btp_core_frame_new, 67	btp_gdb_frame_dup, 79
btp_core_frame_to_json, 67	btp_gdb_frame_free, 79
core_stacktrace.h, 69	btp_gdb_frame_init, 79
btp_core_stacktrace_dup, 69	btp_gdb_frame_new, 80
htn core stacktrace free 69	htn gdb frame parse 80

btp_gdb_frame_parse_address_in_function,	btp_gdb_thread_parse, 95
80	btp_gdb_thread_parse_funs, 95
btp_gdb_frame_parse_file_location, 80	btp_gdb_thread_quality, 95
btp_gdb_frame_parse_frame_start, 81	btp_gdb_thread_quality_counts, 96
btp_gdb_frame_parse_function_call, 81	btp_gdb_thread_remove_frame, 96
btp_gdb_frame_parse_function_name, 81	btp_gdb_thread_remove_frames_above, 96
btp_gdb_frame_parse_function_name_braces.	± <del>=</del>
82	btp_gdb_thread_skip_lwp, 96
btp_gdb_frame_parse_function_name_chunk,	1
82	koops_frame.h, 98
btp_gdb_frame_parse_function_name	btp_koops_frame_append, 98
template, 82	btp_koops_frame_cmp, 98
btp_gdb_frame_parse_header, 82	btp_koops_frame_dup, 99
btp_gdb_frame_parseadd_operator, 83	btp_koops_frame_free, 99
btp_gdb_frame_remove_func_prefix, 83	btp_koops_frame_init, 99
btp_gdb_frame_skip_function_args, 83	btp_koops_frame_new, 99
gdb_sharedlib.h, 84	btp_koops_frame_to_json, 99
btp_gdb_sharedlib_append, 84	koops_stacktrace.h, 101
btp_gdb_sharedlib_count, 85	btp_koops_stacktrace_dup, 101
btp_gdb_sharedlib_dup, 85	btp_koops_stacktrace_free, 101
btp_gdb_sharedlib_find_address, 85	btp_koops_stacktrace_get_frame_count, 102
btp_gdb_sharedlib_free, 85	btp_koops_stacktrace_init, 102
btp_gdb_sharedlib_init, 85	btp_koops_stacktrace_new, 102
btp_gdb_sharedlib_new, 85	btp_koops_stacktrace_parse, 102
btp_gdb_sharedlib_parse, 85	btp_koops_stacktrace_remove_frame, 102
gdb_stacktrace.h, 87	
btp_gdb_stacktrace_dup, 88	len
btp_gdb_stacktrace_find_crash_thread, 88	btp_strbuf, 59
btp_gdb_stacktrace_find_crash_thread, 88 btp_gdb_stacktrace_free, 88	length
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88	length btp_elf_fde, 38
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88	length btp_elf_fde, 38 library_name
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89	length  btp_elf_fde, 38 library_name  btp_gdb_frame, 40
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89	length  btp_elf_fde, 38 library_name  btp_gdb_frame, 40
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_cmp, 104
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_cmp, 104 btp_location_eat_char, 104
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except_one, 91	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_cmp, 104 btp_location_eat_char, 104 btp_location_eat_char_ext, 104
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except one, 91 btp_gdb_stacktrace_set_libnames, 91	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_cmp, 104 btp_location_eat_char, 104 btp_location_eat_char_ext, 104 btp_location_init, 104
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except_one, 91	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_cmp, 104 btp_location_eat_char, 104 btp_location_eat_char_ext, 104
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except one, 91 btp_gdb_stacktrace_set_libnames, 91	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_cmp, 104 btp_location_eat_char, 104 btp_location_eat_char_ext, 104 btp_location_init, 104 btp_location_init, 104 btp_location_to_string, 105
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except one, 91 btp_gdb_stacktrace_set_libnames, 91 btp_gdb_stacktrace_to_text, 92 gdb_thread.h, 93 btp_gdb_thread_append, 94	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_emp, 104 btp_location_eat_char, 104 btp_location_eat_char, 104 btp_location_init, 104 btp_location_to_string, 105 merge_levels
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except one, 91 btp_gdb_stacktrace_set_libnames, 91 btp_gdb_stacktrace_to_text, 92 gdb_thread.h, 93 btp_gdb_thread_append, 94 btp_gdb_thread_append_to_str, 94	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_cmp, 104 btp_location_eat_char, 104 btp_location_eat_char_ext, 104 btp_location_init, 104 btp_location_init, 104 btp_location_to_string, 105
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except_one, 91 btp_gdb_stacktrace_set_libnames, 91 btp_gdb_stacktrace_to_text, 92 gdb_thread_h, 93 btp_gdb_thread_append, 94 btp_gdb_thread_append_to_str, 94 btp_gdb_thread_cmp, 94	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_emp, 104 btp_location_eat_char, 104 btp_location_eat_char, 104 btp_location_init, 104 btp_location_init, 104 btp_location_to_string, 105  merge_levels btp_dendrogram, 36 message
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except_one, 91 btp_gdb_stacktrace_set_libnames, 91 btp_gdb_stacktrace_to_text, 92 gdb_thread.h, 93 btp_gdb_thread_append, 94 btp_gdb_thread_append_to_str, 94 btp_gdb_thread_dup, 94	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_emp, 104 btp_location_eat_char, 104 btp_location_eat_char_ext, 104 btp_location_init, 104 btp_location_to_string, 105  merge_levels btp_dendrogram, 36 message btp_location, 51
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except_one, 91 btp_gdb_stacktrace_set_libnames, 91 btp_gdb_stacktrace_to_text, 92 gdb_thread.h, 93 btp_gdb_thread_append, 94 btp_gdb_thread_append_to_str, 94 btp_gdb_thread_dup, 94 btp_gdb_thread_format_funs, 94	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_emp, 104 btp_location_eat_char, 104 btp_location_eat_char_ext, 104 btp_location_init, 104 btp_location_to_string, 105  merge_levels btp_dendrogram, 36 message btp_location, 51 metrics.h, 106
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except_one, 91 btp_gdb_stacktrace_set_libnames, 91 btp_gdb_stacktrace_to_text, 92 gdb_thread.h, 93 btp_gdb_thread_append, 94 btp_gdb_thread_append_to_str, 94 btp_gdb_thread_dup, 94 btp_gdb_thread_format_funs, 94 btp_gdb_thread_free, 94	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_emp, 104 btp_location_eat_char, 104 btp_location_eat_char_ext, 104 btp_location_init, 104 btp_location_to_string, 105  merge_levels btp_dendrogram, 36 message btp_location, 51 metrics.h, 106 btp_dist_thread_type, 107
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except one, 91 btp_gdb_stacktrace_set_libnames, 91 btp_gdb_stacktrace_to_text, 92 gdb_thread.h, 93 btp_gdb_thread_append, 94 btp_gdb_thread_append_to_str, 94 btp_gdb_thread_format_funs, 94 btp_gdb_thread_format_funs, 94 btp_gdb_thread_free, 94 btp_gdb_thread_get_frame_count, 95	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_emp, 104 btp_location_eat_char, 104 btp_location_eat_char_ext, 104 btp_location_init, 104 btp_location_to_string, 105  merge_levels btp_dendrogram, 36 message btp_location, 51 metrics.h, 106 btp_dist_thread_type, 107 btp_distances_dup, 107
btp_gdb_stacktrace_free, 88 btp_gdb_stacktrace_get_crash_frame, 88 btp_gdb_stacktrace_get_duplication_hash, 88 btp_gdb_stacktrace_get_optimized_thread, 89 btp_gdb_stacktrace_get_thread_count, 89 btp_gdb_stacktrace_init, 89 btp_gdb_stacktrace_limit_frame_depth, 89 btp_gdb_stacktrace_new, 89 btp_gdb_stacktrace_parse, 90 btp_gdb_stacktrace_parse_header, 90 btp_gdb_stacktrace_quality_complex, 91 btp_gdb_stacktrace_quality_simple, 91 btp_gdb_stacktrace_remove_threads_except_one, 91 btp_gdb_stacktrace_set_libnames, 91 btp_gdb_stacktrace_to_text, 92 gdb_thread.h, 93 btp_gdb_thread_append, 94 btp_gdb_thread_append_to_str, 94 btp_gdb_thread_dup, 94 btp_gdb_thread_format_funs, 94 btp_gdb_thread_free, 94	length btp_elf_fde, 38 library_name btp_gdb_frame, 40 libs btp_gdb_stacktrace, 43 line btp_location, 51 location.h, 103 btp_location_add, 103 btp_location_add_ext, 104 btp_location_emp, 104 btp_location_eat_char, 104 btp_location_eat_char_ext, 104 btp_location_init, 104 btp_location_to_string, 105  merge_levels btp_dendrogram, 36 message btp_location, 51 metrics.h, 106 btp_dist_thread_type, 107

btp_distances_new, 107	btp_strbuf_free, 117
btp_distances_set_distance, 108	btp_strbuf_free_nobuf, 117
btp_gdb_threads_compare, 108	btp_strbuf_grow, 117
module_name	btp_strbuf_init, 117
btp_koops_frame, 48	btp_strbuf_new, 117
modules	btp_strbuf_prepend_str, 117
btp_koops_stacktrace, 49	btp_strbuf_prepend_strf, 118
1- 1- /	btp_strbuf_prepend_strfv, 118
next	symbol_name
btp_core_frame, 30	btp_elf_plt_entry, 39
btp_core_thread, 34	··
btp_gdb_frame, 40	taint_mce
btp_gdb_thread, 44	btp_koops_stacktrace, 49
normalize.h, 109	taint_module_proprietary
btp_gdb_normalize_optimize_thread, 109	btp_koops_stacktrace, 50
btp_glibc_thread_find_exit_frame, 109	taint_page_release
btp_normalize_gdb_paired_unknown	btp_koops_stacktrace, 50
function_names, 109	04p_100sps_54404112400, 00
number	unstrip.h, 119
	utils.h, 120
btp_gdb_frame, 41	btp_asprintf, 121
nuthon from h 110	btp_bin2hex, 121
python_frame.h, 110	btp_debug_parser, 125
btp_python_frame_dup, 110	btp_file_to_string, 121
btp_python_frame_free, 110	btp_malloc, 121
btp_python_frame_init, 111	btp_mallocz, 121
btp_python_frame_new, 111	btp_parse_char_cspan, 121
python_stacktrace.h, 112	btp_parse_char_limited, 122
btp_python_stacktrace_dup, 112	btp_parse_char_span, 122
btp_python_stacktrace_free, 112	btp_parse_digit, 122
btp_python_stacktrace_init, 112	btp_parse_hexadecimal_0xuint64, 122
btp_python_stacktrace_new, 113	btp_parse_hexadecimal_uint64, 122
	btp_parse_string, 122
reliable	btp_parse_uint32, 122
btp_koops_frame, 48	btp_realloc, 123
rpm.h, 114	btp_skip_char, 123
btp_rpm_package_append, 114	
	btp_skip_char_cspan, 123
sha1.h, 115	btp_skip_char_limited, 123
signal	btp_skip_char_sequence, 123
btp_core_stacktrace, 32	btp_skip_char_span, 123
signal_handler_called	btp_skip_char_span_location, 123
btp_gdb_frame, 41	btp_skip_hexadecimal_0xuint, 124
source_file	btp_skip_hexadecimal_uint, 124
btp_gdb_frame, 41	btp_skip_string, 124
source_line	btp_skip_uint, 124
btp_gdb_frame, 41	btp_strchr_location, 124
start_address	btp_stremp0, 125
btp_elf_fde, 38	btp_strdup, 125
strbuf.h, 116	btp_strndup, 125
btp_strbuf_append_char, 116	btp_strspn_location, 125
btp_strbuf_append_str, 116	btp_strstr_location, 125
btp_strbuf_append_strf, 116	btp_vasprintf, 125
btp_strbuf_append_strfv, 117	
btp_strbuf_clear, 117	