

# Btparser

A program failure analysis library

Karel Klíč

September 7, 2012



# Contents

<b>1</b>	<b>Overview</b>	<b>9</b>
<b>I</b>	<b>Concepts</b>	<b>11</b>
<b>2</b>	<b>Stack Trace Normalization</b>	<b>13</b>
<b>3</b>	<b>Stack Trace Clustering</b>	<b>15</b>
<b>4</b>	<b>Core Dump Failure Analysis</b>	<b>17</b>
<b>5</b>	<b>Wishlist</b>	<b>19</b>
<b>II</b>	<b>Implementation</b>	<b>21</b>
<b>6</b>	<b>Overview</b>	<b>23</b>
<b>7</b>	<b>Data Structure Index</b>	<b>25</b>
7.1	Data Structures . . . . .	25
7.2	File List . . . . .	25
<b>8</b>	<b>Data Structure Documentation</b>	<b>27</b>
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
8.4	btp_core_stacktrace Struct Reference . . . . .	32

8.4.1	Detailed Description . . . . .	32
8.5	btp_core_thread Struct Reference . . . . .	33
8.5.1	Detailed Description . . . . .	33
8.5.2	Field Documentation . . . . .	33
8.6	btp_dendrogram Struct Reference . . . . .	34
8.6.1	Detailed Description . . . . .	34
8.6.2	Field Documentation . . . . .	34
8.7	btp_disasm_state Struct Reference . . . . .	35
8.7.1	Detailed Description . . . . .	35
8.8	btp_distances Struct Reference . . . . .	36
8.8.1	Detailed Description . . . . .	36
8.9	btp_elf_frame_description_entry Struct Reference . . . . .	37
8.9.1	Detailed Description . . . . .	37
8.9.2	Field Documentation . . . . .	37
8.10	btp_elf_plt_entry Struct Reference . . . . .	38
8.10.1	Detailed Description . . . . .	38
8.10.2	Field Documentation . . . . .	38
8.11	btp_gdb_frame Struct Reference . . . . .	39
8.11.1	Detailed Description . . . . .	39
8.11.2	Field Documentation . . . . .	39
8.12	btp_gdb_sharedlib Struct Reference . . . . .	41
8.12.1	Detailed Description . . . . .	41
8.13	btp_gdb_stacktrace Struct Reference . . . . .	42
8.13.1	Detailed Description . . . . .	42
8.13.2	Field Documentation . . . . .	42
8.14	btp_gdb_thread Struct Reference . . . . .	43
8.14.1	Detailed Description . . . . .	43
8.14.2	Field Documentation . . . . .	43
8.15	btp_koops_frame Struct Reference . . . . .	44
8.16	btp_koops_stacktrace Struct Reference . . . . .	45
8.17	btp_location Struct Reference . . . . .	46
8.17.1	Detailed Description . . . . .	46
8.17.2	Field Documentation . . . . .	46
8.18	btp_python_frame Struct Reference . . . . .	47
8.19	btp_python_stacktrace Struct Reference . . . . .	48
8.20	btp_sha1_state Struct Reference . . . . .	49

8.20.1 Detailed Description . . . . .	49
8.21 btp_strbuf Struct Reference . . . . .	50
8.21.1 Detailed Description . . . . .	50
8.21.2 Field Documentation . . . . .	50
8.22 btp_unstrip_entry Struct Reference . . . . .	51
8.22.1 Detailed Description . . . . .	51
<b>9 File Documentation</b>	<b>53</b>
9.1 callgraph.h File Reference . . . . .	53
9.1.1 Detailed Description . . . . .	53
9.1.2 Function Documentation . . . . .	54
9.2 cluster.h File Reference . . . . .	55
9.2.1 Detailed Description . . . . .	55
9.2.2 Function Documentation . . . . .	55
9.3 core_fingerprint.h File Reference . . . . .	57
9.3.1 Detailed Description . . . . .	57
9.4 core_frame.h File Reference . . . . .	58
9.4.1 Detailed Description . . . . .	58
9.4.2 Function Documentation . . . . .	58
9.5 core_stacktrace.h File Reference . . . . .	61
9.5.1 Detailed Description . . . . .	61
9.5.2 Function Documentation . . . . .	61
9.6 core_thread.h File Reference . . . . .	63
9.6.1 Detailed Description . . . . .	63
9.6.2 Function Documentation . . . . .	63
9.7 disassembler.h File Reference . . . . .	65
9.7.1 Detailed Description . . . . .	65
9.7.2 Function Documentation . . . . .	65
9.8 elves.h File Reference . . . . .	66
9.8.1 Detailed Description . . . . .	66
9.8.2 Function Documentation . . . . .	66
9.9 gdb_frame.h File Reference . . . . .	68
9.9.1 Detailed Description . . . . .	69
9.9.2 Function Documentation . . . . .	69
9.10 gdb_sharedlib.h File Reference . . . . .	76
9.10.1 Detailed Description . . . . .	76
9.10.2 Function Documentation . . . . .	76

9.11	<code>gdb_stacktrace.h</code> File Reference . . . . .	79
9.11.1	Detailed Description . . . . .	79
9.11.2	Function Documentation . . . . .	80
9.12	<code>gdb_thread.h</code> File Reference . . . . .	85
9.12.1	Detailed Description . . . . .	85
9.12.2	Function Documentation . . . . .	86
9.13	<code>koops_frame.h</code> File Reference . . . . .	90
9.13.1	Detailed Description . . . . .	90
9.13.2	Function Documentation . . . . .	90
9.14	<code>koops_stacktrace.h</code> File Reference . . . . .	92
9.14.1	Detailed Description . . . . .	92
9.14.2	Function Documentation . . . . .	92
9.15	<code>location.h</code> File Reference . . . . .	94
9.15.1	Detailed Description . . . . .	94
9.15.2	Function Documentation . . . . .	94
9.16	<code>metrics.h</code> File Reference . . . . .	97
9.16.1	Detailed Description . . . . .	98
9.16.2	Typedef Documentation . . . . .	98
9.16.3	Function Documentation . . . . .	98
9.17	<code>normalize.h</code> File Reference . . . . .	100
9.17.1	Detailed Description . . . . .	100
9.17.2	Function Documentation . . . . .	100
9.18	<code>python_frame.h</code> File Reference . . . . .	101
9.18.1	Detailed Description . . . . .	101
9.19	<code>python_stacktrace.h</code> File Reference . . . . .	102
9.19.1	Detailed Description . . . . .	102
9.20	<code>sha1.h</code> File Reference . . . . .	103
9.20.1	Detailed Description . . . . .	103
9.21	<code>strbuf.h</code> File Reference . . . . .	104
9.21.1	Detailed Description . . . . .	104
9.21.2	Function Documentation . . . . .	104
9.22	<code>unstrip.h</code> File Reference . . . . .	107
9.22.1	Detailed Description . . . . .	107
9.23	<code>utils.h</code> File Reference . . . . .	108
9.23.1	Detailed Description . . . . .	109
9.23.2	Function Documentation . . . . .	109

<b>CONTENTS</b>	<b>7</b>
9.23.3 Variable Documentation . . . . .	113
<b>10 Known Bugs</b>	<b>115</b>
<b>11 Wishlist</b>	<b>117</b>
<b>Index</b>	<b>118</b>





# Chapter 1

## 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. Information 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.



## **Part I**

# **Concepts**



## **Chapter 2**

# **Stack Trace Normalization**



## **Chapter 3**

# **Stack Trace Clustering**





## **Chapter 4**

# **Core Dump Failure Analysis**



## **Chapter 5**

### **Wishlist**

Security Impact.

ABI compatibility check.

Collecting environment data.



## **Part II**

# **Implementation**



## **Chapter 6**

### **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.





# Chapter 7

## 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 ) . . . . .	33
btp_dendrogram (A dendrogram created by clustering ) . . . . .	34
btp_disasm_state (Internal state of a disassembler ) . . . . .	35
btp_distances (A distance matrix of stack trace threads ) . . . . .	36
btp_elf_frame_description_entry (A single item of the .eh_frame section present in ELF binaries )	37
btp_elf_plt_entry (A single item of the Procedure Linkage Table present in ELF binaries ) . . . .	38
btp_gdb_frame (A function call of a GDB-produced stack trace ) . . . . .	39
btp_gdb_sharedlib (A shared library memory location as reported by GDB ) . . . . .	41
btp_gdb_stacktrace (A stack trace produced by GDB ) . . . . .	42
btp_gdb_thread (A thread of execution of a GDB-produced stack trace ) . . . . .	43
btp_koops_frame . . . . .	44
btp_koops_stacktrace . . . . .	45
btp_location (A location of a parser in the input stream ) . . . . .	46
btp_python_frame . . . . .	47
btp_python_stacktrace . . . . .	48
btp_sha1_state (Internal state of a SHA-1 hash algorithm ) . . . . .	49
btp_strbuf (A resizable string buffer ) . . . . .	50
btp_unstrip_entry (Core dump memory layout as reported by the unstrip utility ) . . . . .	51

### 7.2 File List

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

callgraph.h (Calling relationships between subroutines ) . . . . .	53
cluster.h (Clustering for stack trace threads ) . . . . .	55
core_fingerprint.h (Fingerprint algorithm for core stack traces ) . . . . .	57
core_frame.h (Single frame of core stack trace thread ) . . . . .	58
core_stacktrace.h (A stack trace of a core dump ) . . . . .	61

core_thread.h (Single thread of execution of a core stack trace ) . . . . .	63
disassembler.h (BFD-based function disassembler ) . . . . .	65
elves.h (Loading PLT and FDEs from ELF binaries ) . . . . .	66
gdb_frame.h (Single frame of GDB stack trace thread ) . . . . .	68
gdb_sharedlib.h (Shared library information as produced by GDB ) . . . . .	76
gdb_stacktrace.h (Stack trace as produced by GDB ) . . . . .	79
gdb_thread.h (Single thread of execution of GDB stack trace ) . . . . .	85
koops_frame.h (Kernel oops stack frame ) . . . . .	90
koops_stacktrace.h (Kernel oops stack trace structure and related algorithms ) . . . . .	92
location.h (Parser location in input file ) . . . . .	94
metrics.h (Distance between stack trace threads ) . . . . .	97
normalize.h (Normalization of stack traces ) . . . . .	100
python_frame.h (Python frame structure and related algorithms ) . . . . .	101
python_stacktrace.h (Python stack trace structure and related algorithms ) . . . . .	102
sha1.h (An implementation of SHA-1 cryptographic hash function ) . . . . .	103
strbuf.h (A string buffer structure and related algorithms ) . . . . .	104
unstrip.h (Parser for the output of the unstrip utility ) . . . . .	107
utils.h (Various utility functions, macros and variables that do not fit elsewhere ) . . . . .	108

# Chapter 8

## 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.

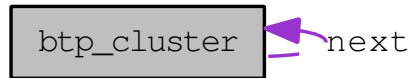
The documentation for this struct was generated from the following file:

- `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.

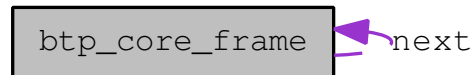
The documentation for this struct was generated from the following file:

- `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.

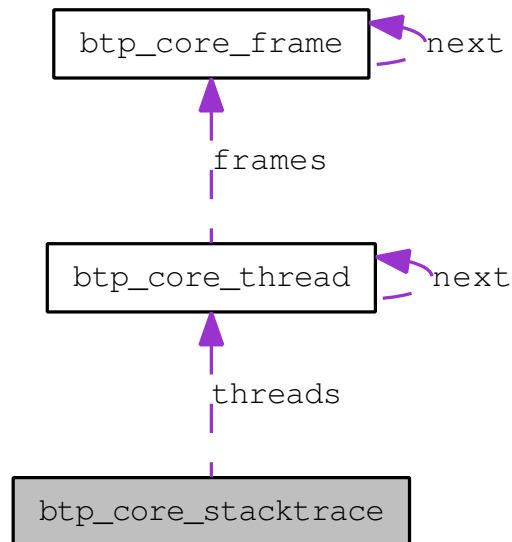
The documentation for this struct was generated from the following file:

- `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

- `struct btp_core_thread * threads`

#### 8.4.1 Detailed Description

A stack trace of a core dump.

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

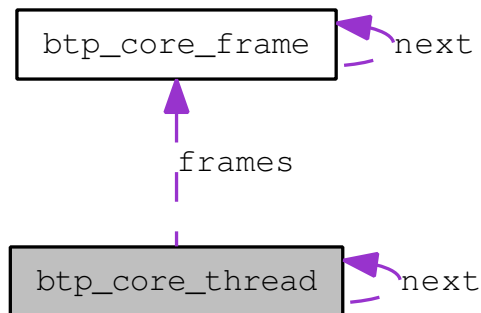
- `core_stacktrace.h`



## 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_dendrogram` Struct Reference

A dendrogram created by clustering.

```
#include <cluster.h>
```

### Data Fields

- `int size`
- `int * order`
- `float * merge_levels`

#### 8.6.1 Detailed Description

A dendrogram created by clustering.

#### 8.6.2 Field Documentation

##### 8.6.2.1 `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`

## 8.7 btp\_disasm\_state Struct Reference

Internal state of a disassembler.

```
#include <disassembler.h>
```

### Data Fields

- bfd \* **bfd\_file**
- disassembler\_ftype **disassembler**
- struct disassemble\_info **info**
- char \* **error\_message**

### 8.7.1 Detailed Description

Internal state of a disassembler.

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

- disassembler.h

## 8.8 `btp_distances` Struct Reference

A distance matrix of stack trace threads.

```
#include <metrics.h>
```

### Data Fields

- `int 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_frame_description_entry` Struct Reference

A single item of the `.eh_frame` section present in ELF binaries.

`#include <elves.h>` Collaboration diagram for `btp_elf_frame_description_entry`:



### Data Fields

- `uint64_t start_address`
- `uint64_t length`
- `struct btp_elf_frame_description_entry * next`

#### 8.9.1 Detailed Description

A single item of the `.eh_frame` section present in ELF binaries.

#### 8.9.2 Field Documentation

##### 8.9.2.1 `uint64_t btp_elf_frame_description_entry::length`

Length of the function in bytes.

##### 8.9.2.2 `uint64_t btp_elf_frame_description_entry::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 `char* btp_elf_plt_entry::symbol_name`

Symbol name corresponding to the address.

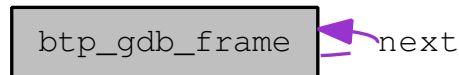
The documentation for this struct was generated from the following file:

- `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
- unsigned number
- char \* source\_file
- unsigned 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 unsigned `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 unsigned `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.

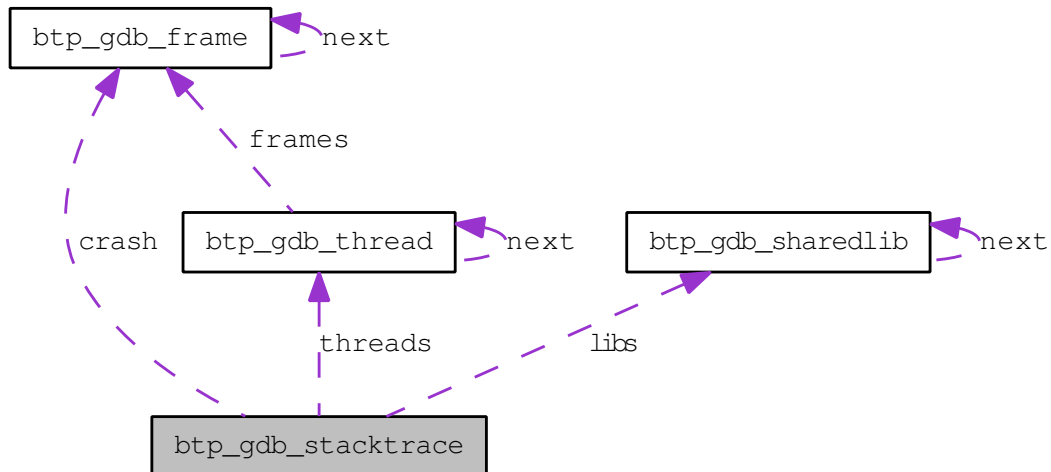
The documentation for this struct was generated from the following file:

- `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.

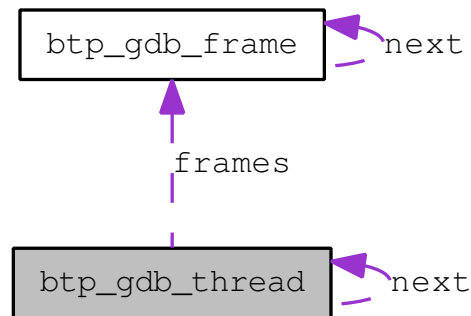
The documentation for this struct was generated from the following file:

- 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

- unsigned **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]

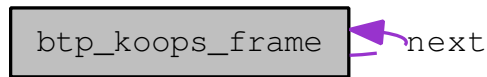
A sibling thread, or NULL if this is the last thread in a stacktrace.

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

- gdb\_thread.h

## 8.15 btp\_koops\_frame Struct Reference

Collaboration diagram for btp\_koops\_frame:



### Data Fields

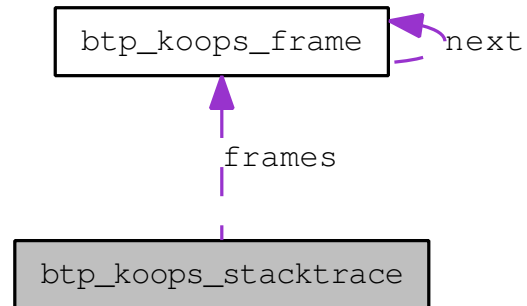
- `uint64_t` **address**
- `bool` **reliable**
- `char *` **name**
- `uint64_t` **offset**
- `uint64_t` **len**
- `char *` **module**
- `struct btp_koops_frame *` **next**

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

- `koops_frame.h`

## 8.16 btp\_koops\_stacktrace Struct Reference

Collaboration diagram for btp\_koops\_stacktrace:



### Data Fields

- `char * version`  
*Version of the kernel.*
- `char * taint`
- `struct btp_koops_frame * frames`  
*Call trace.*

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

- `koops_stacktrace.h`

## 8.17 `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`

#### 8.17.1 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.

#### 8.17.2 Field Documentation

##### 8.17.2.1 `int btp_location::column`

Starts from 0.

##### 8.17.2.2 `int btp_location::line`

Starts from 1.

##### 8.17.2.3 `const char* btp_location::message`

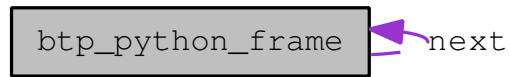
Error message related to the line and column. Do not release the memory this pointer points to.

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

- `location.h`

## 8.18 btp\_python\_frame Struct Reference

Collaboration diagram for btp\_python\_frame:



### Data Fields

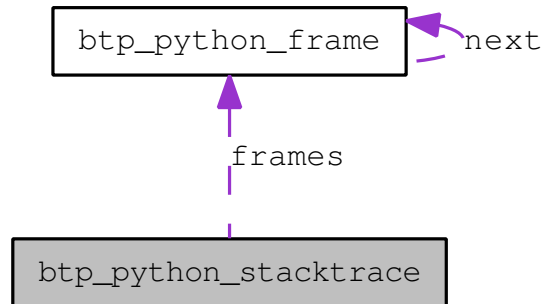
- `char * file_name`
- `uint32_t file_line`
- `bool is_module`
- `char * function_name`
- `char * line`
- `struct btp_python_frame * next`

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

- `python_frame.h`

## 8.19 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`



## 8.20 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]

### 8.20.1 Detailed Description

Internal state of a SHA-1 hash algorithm.

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

- `sha1.h`

## 8.21 `btp_strbuf` Struct Reference

A resizable string buffer.

```
#include <strbuf.h>
```

### Data Fields

- `int alloc`
- `int len`
- `char * buf`

### 8.21.1 Detailed Description

A resizable string buffer.

### 8.21.2 Field Documentation

#### 8.21.2.1 `int btp_strbuf::alloc`

Size of the allocated buffer. Always  $> 0$ .

#### 8.21.2.2 `int btp_strbuf::len`

Length of the string, without the ending `.`

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

- `strbuf.h`

## 8.22 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`

### 8.22.1 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`



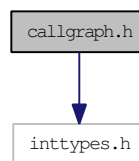
# Chapter 9

## File Documentation

### 9.1 callgraph.h File Reference

Calling relationships between subroutines. `#include <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_frame_description_entry` \*`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_frame_description_entry` \*`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.

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_frame_description_entry * 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()`.

### 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:**

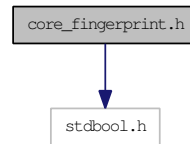
*distances* Distances between the objects. The structure is not modified by calling this function.



## 9.3 core\_fingerprint.h File Reference

Fingerprint algorithm for core stack traces. `#include <stdbool.h>`

Include dependency graph for core\_fingerprint.h:



### Functions

- **bool btp\_core\_fingerprint\_generate** (struct btp\_core\_stacktrace \*stacktrace, char \*\*error\_message)
- **bool btp\_core\_fingerprint\_generate\_for\_binary** (struct btp\_core\_thread \*thread, const char \*binary\_path, char \*\*error\_message)

#### 9.3.1 Detailed Description

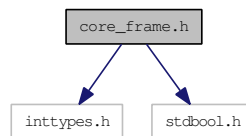
Fingerprint algorithm for core stack traces.

## 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)`
- void `btp_core_frame_append_to_str (struct btp_core_frame *frame, struct btp_strbuf *dest)`

#### 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 void `btp_core_frame_append_to_str` (struct `btp_core_frame` \* *frame*, struct `btp_strbuf` \* *dest*)**

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.

**9.4.2.3 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*.

**9.4.2.4 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 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()`.

**9.4.2.5 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.

**9.4.2.6 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.

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

Creates and initializes a new frame structure.

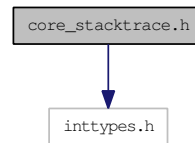
**Returns:**

It never returns NULL. The returned pointer must be released by calling the function `btp_core_frame_free()`.

## 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_text` (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`().

**9.5.2.2 void `btb_core_stacktrace_free` (struct `btb_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 `btb_core_stacktrace_get_thread_count` (struct `btb_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 `btb_core_stacktrace_init` (struct `btb_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 `btb_core_stacktrace`\* `btb_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 `btb_core_stacktrace_free()`.

**9.5.2.6 struct `btb_core_stacktrace`\* `btb_core_stacktrace_parse` (const char \*\* *input*, struct `btb_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 `btb_core_stacktrace_to_text()`.

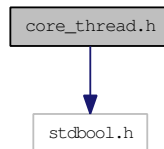
**9.5.2.7 char\* `btb_core_stacktrace_to_text` (struct `btb_core_stacktrace` \* *stacktrace*)**

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)`
- `void btp_core_thread_append_to_str (struct btp_core_thread *thread, struct btp_strbuf *dest)`

#### 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.

**9.6.2.2** `void btp_core_thread_append_to_str (struct btp_core_thread * thread, struct btp_strbuf * dest)`

Appends a textual representation of a thread to a string buffer.

### 9.6.2.3 `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.

### 9.6.2.4 `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.

### 9.6.2.5 `void btp_core_thread_free (struct btp_core_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.

### 9.6.2.6 `int btp_core_thread_get_frame_count (struct btp_core_thread * thread)`

Returns the number of frames in the thread.

### 9.6.2.7 `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.

### 9.6.2.8 `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 disassembler.h File Reference

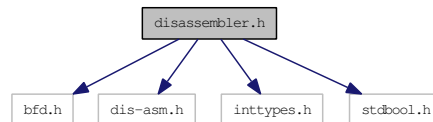
BFD-based function disassembler. `#include <bfd.h>`

`#include <dis-asm.h>`

`#include <inttypes.h>`

`#include <stdbool.h>`

Include dependency graph for `disassembler.h`:



### Data Structures

- struct `btp_disasm_state`  
*Internal state of a disassembler.*

### 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)

#### 9.7.1 Detailed Description

BFD-based function disassembler.

#### 9.7.2 Function Documentation

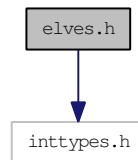
##### 9.7.2.1 char\*\* btp\_disasm\_get\_function\_instructions (struct btp\_disasm\_state \* state, uint64\_t start\_offset, uint64\_t size, char \*\* error\_message)

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

## 9.8 `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_frame_description_entry`  
*A single item 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_frame_description_entry * btp_elf_get_eh_frame (const char *filename, char **error_message)`
- `void btp_elf_eh_frame_free (struct btp_elf_frame_description_entry *entries)`
- `struct btp_elf_frame_description_entry * btp_elf_find_fde_for_address (struct btp_elf_frame_description_entry *eh_frame, uint64_t build_id_offset)`

#### 9.8.1 Detailed Description

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

#### 9.8.2 Function Documentation

- 9.8.2.1** `struct btp_elf_frame_description_entry* 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.

**9.8.2.2** `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:**

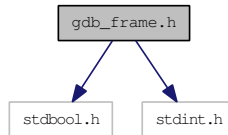
Linked list of PLT entries on success. NULL otherwise.

## 9.9 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)`
- `bool btp_gdb_frame_calls_func_in_file (struct btp_gdb_frame *frame, const char *function_name, const char *source_file)`
- `bool btp_gdb_frame_calls_func_in_file2 (struct btp_gdb_frame *frame, const char *function_name, const char *source_file0, const char *source_file1)`
- `bool btp_gdb_frame_calls_func_in_file3 (struct btp_gdb_frame *frame, const char *function_name, const char *source_file0, const char *source_file1, const char *source_file2)`
- `bool btp_gdb_frame_calls_func_in_file4 (struct btp_gdb_frame *frame, const char *function_name, const char *source_file0, const char *source_file1, const char *source_file2, const char *source_file3)`
- `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, unsigned *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, unsigned *fileline, 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)`

### 9.9.1 Detailed Description

Single frame of GDB stack trace thread.

### 9.9.2 Function Documentation

**9.9.2.1** `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.

**9.9.2.2** `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.

**9.9.2.3** `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.

**9.9.2.4** `bool btp_gdb_frame_calls_func_in_file (struct btp_gdb_frame * frame, const char * function_name, const char * source_file)`

Checks whether the frame represents a call of function with certain function name, which resides in a source file.

#### Parameters:

*source\_file* The frame's source\_file is searched for the source\_file as a substring.

### 9.9.2.5 `bool btp_gdb_frame_calls_func_in_file2 (struct btp_gdb_frame * frame, const char * function_name, const char * source_file0, const char * source_file1)`

Checks whether the frame represents a call of function with certain function name, which resides in one of the source files.

#### Parameters:

*source\_file0* The frame's `source_file` is searched for the `source_file0` as a substring.

#### Returns:

True if the frame corresponds to a function with `function_name`, residing in the `source_file0`, or `source_file1`.

### 9.9.2.6 `bool btp_gdb_frame_calls_func_in_file3 (struct btp_gdb_frame * frame, const char * function_name, const char * source_file0, const char * source_file1, const char * source_file2)`

Checks whether the frame represents a call of function with certain function name, which resides in one of the source files.

#### Parameters:

*source\_file0* The frame's `source_file` is searched for the `source_file0` as a substring.

#### Returns:

True if the frame corresponds to a function with `function_name`, residing in the `source_file0`, `source_file1`, or `source_file2`.

### 9.9.2.7 `bool btp_gdb_frame_calls_func_in_file4 (struct btp_gdb_frame * frame, const char * function_name, const char * source_file0, const char * source_file1, const char * source_file2, const char * source_file3)`

Checks whether the frame represents a call of function with certain function name, which resides in one of the source files.

#### Parameters:

*source\_file0* The frame's `source_file` is searched for the `source_file0` as a substring.

#### Returns:

True if the frame corresponds to a function with `function_name`, residing in the `source_file0`, `source_file1`, `source_file2`, or `source_file3`.

### 9.9.2.8 `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 comparison. 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.

### 9.9.2.9 `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.

### 9.9.2.10 `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().

### 9.9.2.11 `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.

### 9.9.2.12 void `btg_gdb_frame_init` (struct `btg_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.

### 9.9.2.13 struct `btg_gdb_frame`\* `btg_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 `btg_gdb_frame_free()`.

### 9.9.2.14 struct `btg_gdb_frame`\* `btg_gdb_frame_parse` (const char \*\* *input*, struct `btg_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 `btg_gdb_frame_free()`.

#### Parameters:

**location** The caller must provide a pointer to an instance of `btg_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.

### 9.9.2.15 bool `btg_gdb_frame_parse_address_in_function` (const char \*\* *input*, uint64\_t \* *address*, char \*\* *function\_name*, char \*\* *function\_type*, struct `btg_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 `btg_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.



### 9.9.2.16 `bool` `btp_gdb_frame_parse_file_location` (`const char **input`, `char **file`, `unsigned *fileline`, `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.

### 9.9.2.17 `int` `btp_gdb_frame_parse_frame_start` (`const char **input`, `unsigned *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 "
```

### 9.9.2.18 `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 `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.

### 9.9.2.19 `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 `btb_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.

#### 9.9.2.20 `int btb_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.

#### 9.9.2.21 `int btb_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.

#### 9.9.2.22 `int btb_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.

#### 9.9.2.23 `struct btb_gdb_frame* btb_gdb_frame_parse_header (const char ** input, struct btb_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 `btb_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()`.

**9.9.2.24 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.

**9.9.2.25 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.

**9.9.2.26 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:**

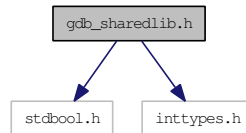
*location* The caller must provide a pointer to an instance of `btp_location` here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns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

## 9.10 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 siblings)`
- `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)`

#### 9.10.1 Detailed Description

Shared library information as produced by GDB.

#### 9.10.2 Function Documentation

##### 9.10.2.1 `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.

**9.10.2.2 int btp\_gdb\_sharedlib\_count (struct btp\_gdb\_sharedlib \* *sharedlib*)**

Returns the number of sharedlibs in the list.

**9.10.2.3 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.

**9.10.2.4 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.

**9.10.2.5 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.

**9.10.2.6 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.

**9.10.2.7 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().

**9.10.2.8 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.

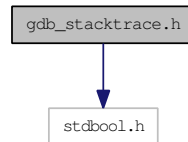
**Returns:**

First element of the list of loaded libraries.

## 9.11 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)`

#### 9.11.1 Detailed Description

Stack trace as produced by GDB.

## 9.11.2 Function Documentation

### 9.11.2.1 `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()`.

### 9.11.2.2 `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.

### 9.11.2.3 `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.

### 9.11.2.4 `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.



### 9.11.2.5 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()`.

### 9.11.2.6 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.

### 9.11.2.7 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.

### 9.11.2.8 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.

### 9.11.2.9 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.

### 9.11.2.10 `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()`.

### 9.11.2.11 `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.

```
struct btp_location location;
btp_location_init(&location);
char *input = "...";
struct btp_gdb_stacktrace *stacktrace;
stacktrace = btp_gdb_stacktrace_parse(input, location);
if (!stacktrace)
{
    fprintf(stderr,
            "Failed to parse the stacktrace.\n"
            "Line %d, column %d: %s\n",
            location.line,
            location.column,
            location.message);
    exit(-1);
}
btp_gdb_stacktrace_free(stacktrace);
```

#### 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()`.

### 9.11.2.12 `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.

```
[New Thread 11919]
[New Thread 11917]
Core was generated by 'evince file:
Program terminated with signal 8, Arithmetic exception.
#0  0x000000322a2362b9 in repeat (image=<value optimized out>,
    mask=<value optimized out>, mask_bits=<value optimized out>)
    at pixman-bits-image.c:145
145     pixman-bits-image.c: No such file or directory.
    in pixman-bits-image.c
```

#### 9.11.2.13 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.

#### 9.11.2.14 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).

#### 9.11.2.15 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.

**9.11.2.16 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.

**9.11.2.17 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.

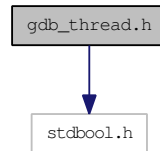
**Returns:**

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

## 9.12 gdb\_thread.h File Reference

Single thread of execution of GDB stack trace. `#include <stdbool.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)

#### 9.12.1 Detailed Description

Single thread of execution of GDB stack trace.

## 9.12.2 Function Documentation

**9.12.2.1** `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.

**9.12.2.2** `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'.

**9.12.2.3** `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.

**9.12.2.4** `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.

**9.12.2.5** `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().

**9.12.2.6 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.

**9.12.2.7 int `btp_gdb_thread_get_frame_count` (struct `btp_gdb_thread` \* *thread*)**

Returns the number of frames in the thread.

**9.12.2.8 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.

**9.12.2.9 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()`.

**9.12.2.10 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()`.

**9.12.2.11 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()`.

**9.12.2.12 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.

**9.12.2.13 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*.

**9.12.2.14 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.

**9.12.2.15 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.

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

Keeps only the top *n* frames in the thread.



**9.12.2.17 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:**

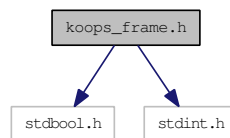
The number of characters parsed from input. 0 if the input does not contain a LWP section.

## 9.13 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

### 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)

#### 9.13.1 Detailed Description

Kernel oops stack frame.

#### 9.13.2 Function Documentation

##### 9.13.2.1 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().

**9.13.2.2 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.

**9.13.2.3 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.

**9.13.2.4 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 kops\_stacktrace.h File Reference

Kernel oops stack trace structure and related algorithms.

### Data Structures

- struct btp\_kops\_stacktrace

### Functions

- struct btp\_kops\_stacktrace \* btp\_kops\_stacktrace\_new ()
- void btp\_kops\_stacktrace\_init (struct btp\_kops\_stacktrace \*stacktrace)
- void btp\_kops\_stacktrace\_free (struct btp\_kops\_stacktrace \*stacktrace)
- struct btp\_kops\_stacktrace \* btp\_kops\_stacktrace\_dup (struct btp\_kops\_stacktrace \*stacktrace)
- struct btp\_kops\_stacktrace \* btp\_kops\_stacktrace\_parse (const char \*\*input, struct btp\_location \*location)

#### 9.14.1 Detailed Description

Kernel oops stack trace structure and related algorithms.

#### 9.14.2 Function Documentation

##### 9.14.2.1 struct btp\_kops\_stacktrace\* btp\_kops\_stacktrace\_dup (struct btp\_kops\_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\_kops\_stacktrace\_free().

##### 9.14.2.2 void btp\_kops\_stacktrace\_free (struct btp\_kops\_stacktrace \* stacktrace)

Releases the memory held by the stacktrace.

##### Parameters:

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

##### 9.14.2.3 void btp\_kops\_stacktrace\_init (struct btp\_kops\_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.

#### 9.14.2.4 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().

#### 9.14.2.5 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.

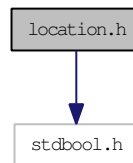
**Parameters:**

*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 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)`

#### 9.15.1 Detailed Description

Parser location in input file.

#### 9.15.2 Function Documentation

##### 9.15.2.1 `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.

#### 9.15.2.2 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 first 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.

#### 9.15.2.3 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.

#### 9.15.2.4 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.

#### 9.15.2.5 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.

**9.15.2.6 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.

**9.15.2.7 char\* btp\_location\_to\_string (struct btp\_location \* *location*)**

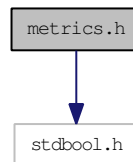
Creates a string representation of location. User must delete the returned string using free().



## 9.16 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)

### 9.16.1 Detailed Description

Distance between stack trace threads.

### 9.16.2 Typedef Documentation

#### 9.16.2.1 `typedef float(* btp_dist_thread_type)(struct btp_gdb_thread *, struct btp_gdb_thread *)`

A function which compares two threads.

### 9.16.3 Function Documentation

#### 9.16.3.1 `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.

#### 9.16.3.2 `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.

#### 9.16.3.3 `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.

**9.16.3.4 struct `btpr_distances*` `btpr_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 `btpr_distances_free()`.

**9.16.3.5 void `btpr_distances_set_distance` (struct `btpr_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.

**9.16.3.6 struct `btpr_distances*` `btpr_gdb_threads_compare` (struct `btpr_gdb_thread **`*threads*, int *m*, int *n*, `btpr_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.

## 9.17 normalize.h File Reference

Normalization of stack traces.

### Functions

- void **btb\_normalize\_thread** (struct btb\_gdb\_thread \*thread)
- void **btb\_normalize\_stacktrace** (struct btb\_gdb\_stacktrace \*stacktrace)
- void **btb\_normalize\_dbus\_thread** (struct btb\_gdb\_thread \*thread)
- void **btb\_normalize\_gdk\_thread** (struct btb\_gdb\_thread \*thread)
- void **btb\_normalize\_gtk\_thread** (struct btb\_gdb\_thread \*thread)
- void **btb\_normalize\_glib\_thread** (struct btb\_gdb\_thread \*thread)
- struct btb\_gdb\_frame \* btb\_glibc\_thread\_find\_exit\_frame (struct btb\_gdb\_thread \*thread)
- void **btb\_normalize\_glibc\_thread** (struct btb\_gdb\_thread \*thread)
- void **btb\_normalize\_libstdcpp\_thread** (struct btb\_gdb\_thread \*thread)
- void **btb\_normalize\_linux\_thread** (struct btb\_gdb\_thread \*thread)
- void **btb\_normalize\_xorg\_thread** (struct btb\_gdb\_thread \*thread)
- void btb\_normalize\_paired\_unknown\_function\_names (struct btb\_gdb\_thread \*thread1, struct btb\_gdb\_thread \*thread2)
- void btb\_normalize\_optimize\_thread (struct btb\_gdb\_thread \*thread)

### 9.17.1 Detailed Description

Normalization of stack traces.

### 9.17.2 Function Documentation

#### 9.17.2.1 struct btb\_gdb\_frame\* btb\_glibc\_thread\_find\_exit\_frame (struct btb\_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.

#### 9.17.2.2 void btb\_normalize\_optimize\_thread (struct btb\_gdb\_thread \* thread)

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

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

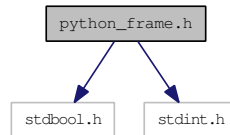
Renames unknown function names ("??") that are between the same function names to be treated as similar in later comparison. Leaves unpair unknown functions unchanged

## 9.18 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`

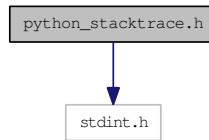
#### 9.18.1 Detailed Description

Python frame structure and related algorithms.

## 9.19 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`

#### 9.19.1 Detailed Description

Python stack trace structure and related algorithms.

## 9.20 sha1.h File Reference

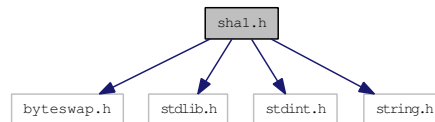
An implementation of SHA-1 cryptographic hash function. `#include <byteswap.h>`

`#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)
- char \* `btp_bin2hex` (char \*dst, const char \*str, int count)

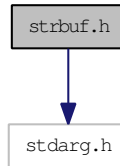
#### 9.20.1 Detailed Description

An implementation of SHA-1 cryptographic hash function.

## 9.21 strbuf.h File Reference

A string buffer structure and related algorithms. `#include <strbuf.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)`

#### 9.21.1 Detailed Description

A string buffer structure and related algorithms.

#### 9.21.2 Function Documentation

##### 9.21.2.1 `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.

##### 9.21.2.2 `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.



**9.21.2.3 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.

**9.21.2.4 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.

**9.21.2.5 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.

**9.21.2.6 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.

**9.21.2.7 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().

**9.21.2.8 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.

**9.21.2.9 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 overwritten. This is useful for initializing a strbuf structure placed on the stack.

**9.21.2.10 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().

**9.21.2.11** `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.

**9.21.2.12** `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.

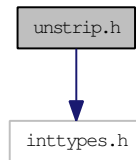
**9.21.2.13** `struct btp_strbuf* btp_strbuf_prepend_strfv (struct btp_strbuf * strbuf, const char * format, va_list p)` [read]

Same as `btp_strbuf_prepend_strf` except that *va\_list* is used instead of variable number of arguments.

## 9.22 unstrip.h File Reference

Parser for the output of the unstrip utility. `#include <inttypes.h>`

Include dependency graph for unstrip.h:



### Data Structures

- struct `btb_unstrip_entry`

*Core dump memory layout as reported by the unstrip utility.*

### Functions

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

#### 9.22.1 Detailed Description

Parser for the output of the unstrip utility.

## 9.23 utils.h File Reference

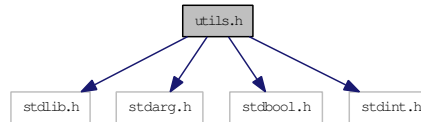
Various utility functions, macros and variables that do not fit elsewhere. `#include <stdlib.h>`

`#include <stdarg.h>`

`#include <stdbool.h>`

`#include <stdint.h>`

Include dependency graph for `utils.h`:



### Defines

- `#define BTP_lower "abcdefghijklmnopqrstuvwxyz"`
- `#define BTP_upper "ABCDEFGHIJKLMNOPQRSTUVWXYZ"`
- `#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_stchr_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)`
- `bool btp_parse_char_cspan (const char **input, const char *reject, char **result)`
- `int btp_skip_string (const char **input, const char *string)`
- `bool btp_parse_string (const char **input, const char *string, char **result)`

- char `btp_parse_digit` (const char \*\*input)
- int `btp_skip_unsigned_integer` (const char \*\*input)
- int `btp_parse_unsigned_integer` (const char \*\*input, unsigned \*result)
- int `btp_skip_hexadecimal_number` (const char \*\*input)
- int `btp_parse_hexadecimal_number` (const char \*\*input, uint64\_t \*result)
- char \* `btp_skip_whitespace` (const char \*s)
- char \* `btp_skip_non_whitespace` (const char \*s)

## Variables

- bool `btp_debug_parser`

### 9.23.1 Detailed Description

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

### 9.23.2 Function Documentation

#### 9.23.2.1 char\* `btp_asprintf` (const char \**format*, ...)

Never returns NULL.

#### 9.23.2.2 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.

#### 9.23.2.3 void\* `btp_malloc` (size\_t *size*)

Never returns NULL.

#### 9.23.2.4 void\* `btp_mallocz` (size\_t *size*)

Never returns NULL.

#### 9.23.2.5 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.

**9.23.2.6 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.

**9.23.2.7 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.

**9.23.2.8 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.

**9.23.2.9 int btp\_parse\_hexadecimal\_number (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.

**9.23.2.10 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.

**9.23.2.11 int btp\_parse\_unsigned\_integer (const char \*\* *input*, unsigned \* *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.

**9.23.2.12 void\* btp\_realloc (void \* *ptr*, size\_t *size*)**

Never returns NULL.

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

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

**9.23.2.14** `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.

**9.23.2.15** `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*.

**9.23.2.16** `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.

**9.23.2.17** `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.

**9.23.2.18** `int btp_skip_hexadecimal_number (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.

**9.23.2.19** `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.

**9.23.2.20 int btp\_skip\_unsigned\_integer (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.

**9.23.2.21 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.

**9.23.2.22 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.

**9.23.2.23 char\* btp\_strdup (const char \* *s*)**

Never returns NULL.

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

Never returns NULL.

**9.23.2.25 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.

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

A strstr() variant providing line and column of the haystack 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.



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

Never returns NULL.

**9.23.3 Variable Documentation****9.23.3.1 bool btp\_debug\_parser**

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



## **Chapter 10**

# **Known Bugs**

Empty.



## **Chapter 11**

### **Wishlist**

Stack trace for kerneloopses, Python, and Java.

# Index

address  
  btp\_core\_frame, 26  
  btp\_elf\_plt\_entry, 34  
  btp\_gdb\_frame, 35  
alloc  
  btp\_strbuf, 46  
  
btp\_asprintf  
  utils.h, 105  
btp\_callgraph, 23  
  callees, 23  
btp\_callgraph\_extend  
  callgraph.h, 50  
btp\_cluster, 25  
btp\_cluster\_free  
  cluster.h, 51  
btp\_cluster\_new  
  cluster.h, 51  
btp\_core\_frame, 26  
  address, 26  
  build\_id, 26  
  fingerprint, 26  
  next, 26  
btp\_core\_frame\_append  
  core\_frame.h, 54  
btp\_core\_frame\_append\_to\_str  
  core\_frame.h, 54  
btp\_core\_frame\_cmp  
  core\_frame.h, 55  
btp\_core\_frame\_dup  
  core\_frame.h, 55  
btp\_core\_frame\_free  
  core\_frame.h, 55  
btp\_core\_frame\_init  
  core\_frame.h, 55  
btp\_core\_frame\_new  
  core\_frame.h, 55  
btp\_core\_stacktrace, 28  
btp\_core\_stacktrace\_dup  
  core\_stacktrace.h, 57  
btp\_core\_stacktrace\_free  
  core\_stacktrace.h, 57  
btp\_core\_stacktrace\_get\_thread\_count  
  core\_stacktrace.h, 58  
btp\_core\_stacktrace\_init  
  core\_stacktrace.h, 58  
btp\_core\_stacktrace\_new  
  core\_stacktrace.h, 58  
btp\_core\_stacktrace\_parse  
  core\_stacktrace.h, 58  
btp\_core\_stacktrace\_to\_text  
  core\_stacktrace.h, 58  
btp\_core\_thread, 29  
  frames, 29  
  next, 29  
btp\_core\_thread\_append  
  core\_thread.h, 59  
btp\_core\_thread\_append\_to\_str  
  core\_thread.h, 59  
btp\_core\_thread\_cmp  
  core\_thread.h, 59  
btp\_core\_thread\_dup  
  core\_thread.h, 60  
btp\_core\_thread\_free  
  core\_thread.h, 60  
btp\_core\_thread\_get\_frame\_count  
  core\_thread.h, 60  
btp\_core\_thread\_init  
  core\_thread.h, 60  
btp\_core\_thread\_new  
  core\_thread.h, 60  
btp\_debug\_parser  
  utils.h, 109  
btp\_dendrogram, 30  
  merge\_levels, 30  
btp\_dendrogram\_cut  
  cluster.h, 51  
btp\_dendrogram\_free  
  cluster.h, 52  
btp\_dendrogram\_new  
  cluster.h, 52  
btp\_disasm\_get\_function\_instructions  
  disassembler.h, 61  
btp\_disasm\_state, 31  
btp\_dist\_thread\_type  
  metrics.h, 94  
btp\_distances, 32  
btp\_distances\_cluster\_objects  
  cluster.h, 52  
btp\_distances\_dup

- metrics.h, 94
- btp\_distances\_free
  - metrics.h, 94
- btp\_distances\_get\_distance
  - metrics.h, 94
- btp\_distances\_new
  - metrics.h, 94
- btp\_distances\_set\_distance
  - metrics.h, 95
- btp\_elf\_frame\_description\_entry, 33
  - length, 33
  - start\_address, 33
- btp\_elf\_get\_eh\_frame
  - elves.h, 62
- btp\_elf\_get\_procedure\_linkage\_table
  - elves.h, 63
- btp\_elf\_plt\_entry, 34
  - address, 34
  - symbol\_name, 34
- btp\_file\_to\_string
  - utils.h, 105
- btp\_gdb\_frame, 35
  - address, 35
  - function\_name, 35
  - function\_type, 35
  - library\_name, 35
  - next, 35
  - number, 36
  - signal\_handler\_called, 36
  - source\_file, 36
  - source\_line, 36
- btp\_gdb\_frame\_append
  - gdb\_frame.h, 65
- btp\_gdb\_frame\_append\_to\_str
  - gdb\_frame.h, 65
- btp\_gdb\_frame\_calls\_func
  - gdb\_frame.h, 65
- btp\_gdb\_frame\_calls\_func\_in\_file
  - gdb\_frame.h, 65
- btp\_gdb\_frame\_calls\_func\_in\_file2
  - gdb\_frame.h, 65
- btp\_gdb\_frame\_calls\_func\_in\_file3
  - gdb\_frame.h, 66
- btp\_gdb\_frame\_calls\_func\_in\_file4
  - gdb\_frame.h, 66
- btp\_gdb\_frame\_cmp
  - gdb\_frame.h, 66
- btp\_gdb\_frame\_cmp\_simple
  - gdb\_frame.h, 67
- btp\_gdb\_frame\_dup
  - gdb\_frame.h, 67
- btp\_gdb\_frame\_free
  - gdb\_frame.h, 67
- btp\_gdb\_frame\_init
  - gdb\_frame.h, 67
- btp\_gdb\_frame\_new
  - gdb\_frame.h, 68
- btp\_gdb\_frame\_parse
  - gdb\_frame.h, 68
- btp\_gdb\_frame\_parse\_address\_in\_function
  - gdb\_frame.h, 68
- btp\_gdb\_frame\_parse\_file\_location
  - gdb\_frame.h, 68
- btp\_gdb\_frame\_parse\_frame\_start
  - gdb\_frame.h, 69
- btp\_gdb\_frame\_parse\_function\_call
  - gdb\_frame.h, 69
- btp\_gdb\_frame\_parse\_function\_name
  - gdb\_frame.h, 69
- btp\_gdb\_frame\_parse\_function\_name\_braces
  - gdb\_frame.h, 70
- btp\_gdb\_frame\_parse\_function\_name\_chunk
  - gdb\_frame.h, 70
- btp\_gdb\_frame\_parse\_function\_name\_template
  - gdb\_frame.h, 70
- btp\_gdb\_frame\_parse\_header
  - gdb\_frame.h, 70
- btp\_gdb\_frame\_parseadd\_operator
  - gdb\_frame.h, 71
- btp\_gdb\_frame\_remove\_func\_prefix
  - gdb\_frame.h, 71
- btp\_gdb\_frame\_skip\_function\_args
  - gdb\_frame.h, 71
- btp\_gdb\_sharedlib, 37
- btp\_gdb\_sharedlib\_append
  - gdb\_sharedlib.h, 72
- btp\_gdb\_sharedlib\_count
  - gdb\_sharedlib.h, 73
- btp\_gdb\_sharedlib\_dup
  - gdb\_sharedlib.h, 73
- btp\_gdb\_sharedlib\_find\_address
  - gdb\_sharedlib.h, 73
- btp\_gdb\_sharedlib\_free
  - gdb\_sharedlib.h, 73
- btp\_gdb\_sharedlib\_init
  - gdb\_sharedlib.h, 73
- btp\_gdb\_sharedlib\_new
  - gdb\_sharedlib.h, 73
- btp\_gdb\_sharedlib\_parse
  - gdb\_sharedlib.h, 73
- btp\_gdb\_stacktrace, 38
  - crash, 38
  - libs, 38
- btp\_gdb\_stacktrace\_dup
  - gdb\_stacktrace.h, 76
- btp\_gdb\_stacktrace\_find\_crash\_thread
  - gdb\_stacktrace.h, 76
- btp\_gdb\_stacktrace\_free

- gdb\_stacktrace.h, 76
- btg\_gdb\_stacktrace\_get\_crash\_frame
  - gdb\_stacktrace.h, 76
- btg\_gdb\_stacktrace\_get\_duplication\_hash
  - gdb\_stacktrace.h, 76
- btg\_gdb\_stacktrace\_get\_optimized\_thread
  - gdb\_stacktrace.h, 77
- btg\_gdb\_stacktrace\_get\_thread\_count
  - gdb\_stacktrace.h, 77
- btg\_gdb\_stacktrace\_init
  - gdb\_stacktrace.h, 77
- btg\_gdb\_stacktrace\_limit\_frame\_depth
  - gdb\_stacktrace.h, 77
- btg\_gdb\_stacktrace\_new
  - gdb\_stacktrace.h, 77
- btg\_gdb\_stacktrace\_parse
  - gdb\_stacktrace.h, 78
- btg\_gdb\_stacktrace\_parse\_header
  - gdb\_stacktrace.h, 78
- btg\_gdb\_stacktrace\_quality\_complex
  - gdb\_stacktrace.h, 79
- btg\_gdb\_stacktrace\_quality\_simple
  - gdb\_stacktrace.h, 79
- btg\_gdb\_stacktrace\_remove\_threads\_except\_one
  - gdb\_stacktrace.h, 79
- btg\_gdb\_stacktrace\_set\_libnames
  - gdb\_stacktrace.h, 79
- btg\_gdb\_stacktrace\_to\_text
  - gdb\_stacktrace.h, 80
- btg\_gdb\_thread, 39
  - frames, 39
  - next, 39
- btg\_gdb\_thread\_append
  - gdb\_thread.h, 82
- btg\_gdb\_thread\_append\_to\_str
  - gdb\_thread.h, 82
- btg\_gdb\_thread\_cmp
  - gdb\_thread.h, 82
- btg\_gdb\_thread\_dup
  - gdb\_thread.h, 82
- btg\_gdb\_thread\_format\_funs
  - gdb\_thread.h, 82
- btg\_gdb\_thread\_free
  - gdb\_thread.h, 82
- btg\_gdb\_thread\_get\_frame\_count
  - gdb\_thread.h, 83
- btg\_gdb\_thread\_init
  - gdb\_thread.h, 83
- btg\_gdb\_thread\_new
  - gdb\_thread.h, 83
- btg\_gdb\_thread\_parse
  - gdb\_thread.h, 83
- btg\_gdb\_thread\_parse\_funs
  - gdb\_thread.h, 83
- btg\_gdb\_thread\_quality
  - gdb\_thread.h, 83
- btg\_gdb\_thread\_quality\_counts
  - gdb\_thread.h, 84
- btg\_gdb\_thread\_remove\_frame
  - gdb\_thread.h, 84
- btg\_gdb\_thread\_remove\_frames\_above
  - gdb\_thread.h, 84
- btg\_gdb\_thread\_remove\_frames\_below\_n
  - gdb\_thread.h, 84
- btg\_gdb\_thread\_skip\_lwp
  - gdb\_thread.h, 84
- btg\_gdb\_threads\_compare
  - metrics.h, 95
- btg\_glibc\_thread\_find\_exit\_frame
  - normalize.h, 96
- btg\_koops\_frame, 40
- btg\_koops\_frame\_dup
  - koops\_frame.h, 86
- btg\_koops\_frame\_free
  - koops\_frame.h, 86
- btg\_koops\_frame\_init
  - koops\_frame.h, 87
- btg\_koops\_frame\_new
  - koops\_frame.h, 87
- btg\_koops\_stacktrace, 41
- btg\_koops\_stacktrace\_dup
  - koops\_stacktrace.h, 88
- btg\_koops\_stacktrace\_free
  - koops\_stacktrace.h, 88
- btg\_koops\_stacktrace\_init
  - koops\_stacktrace.h, 88
- btg\_koops\_stacktrace\_new
  - koops\_stacktrace.h, 88
- btg\_koops\_stacktrace\_parse
  - koops\_stacktrace.h, 89
- btg\_location, 42
  - column, 42
  - line, 42
  - message, 42
- btg\_location\_add
  - location.h, 90
- btg\_location\_add\_ext
  - location.h, 91
- btg\_location\_cmp
  - location.h, 91
- btg\_location\_eat\_char
  - location.h, 91
- btg\_location\_eat\_char\_ext
  - location.h, 91
- btg\_location\_init
  - location.h, 91
- btg\_location\_to\_string
  - location.h, 92



- btp\_malloc
  - utils.h, 105
- btp\_mallocz
  - utils.h, 105
- btp\_normalize\_optimize\_thread
  - normalize.h, 96
- btp\_normalize\_paired\_unknown\_function\_names
  - normalize.h, 96
- btp\_parse\_char\_cspan
  - utils.h, 105
- btp\_parse\_char\_limited
  - utils.h, 105
- btp\_parse\_char\_span
  - utils.h, 106
- btp\_parse\_digit
  - utils.h, 106
- btp\_parse\_hexadecimal\_number
  - utils.h, 106
- btp\_parse\_string
  - utils.h, 106
- btp\_parse\_unsigned\_integer
  - utils.h, 106
- btp\_python\_frame, 43
- btp\_python\_stacktrace, 44
- btp\_realloc
  - utils.h, 106
- btp\_sha1\_state, 45
- btp\_skip\_char
  - utils.h, 106
- btp\_skip\_char\_limited
  - utils.h, 106
- btp\_skip\_char\_sequence
  - utils.h, 107
- btp\_skip\_char\_span
  - utils.h, 107
- btp\_skip\_char\_span\_location
  - utils.h, 107
- btp\_skip\_hexadecimal\_number
  - utils.h, 107
- btp\_skip\_string
  - utils.h, 107
- btp\_skip\_unsigned\_integer
  - utils.h, 107
- btp\_strbuf, 46
  - alloc, 46
  - len, 46
- btp\_strbuf\_append\_char
  - strbuf.h, 100
- btp\_strbuf\_append\_str
  - strbuf.h, 100
- btp\_strbuf\_append\_strf
  - strbuf.h, 100
- btp\_strbuf\_append\_strfv
  - strbuf.h, 101
- btp\_strbuf\_clear
  - strbuf.h, 101
- btp\_strbuf\_free
  - strbuf.h, 101
- btp\_strbuf\_free\_nobuf
  - strbuf.h, 101
- btp\_strbuf\_grow
  - strbuf.h, 101
- btp\_strbuf\_init
  - strbuf.h, 101
- btp\_strbuf\_new
  - strbuf.h, 101
- btp\_strbuf\_prepend\_str
  - strbuf.h, 101
- btp\_strbuf\_prepend\_strf
  - strbuf.h, 102
- btp\_strbuf\_prepend\_strfv
  - strbuf.h, 102
- btp\_strchr\_location
  - utils.h, 108
- btp\_strcmp0
  - utils.h, 108
- btp\_strdup
  - utils.h, 108
- btp\_strndup
  - utils.h, 108
- btp\_strspn\_location
  - utils.h, 108
- btp\_strstr\_location
  - utils.h, 108
- btp\_unstrip\_entry, 47
- btp\_vasprintf
  - utils.h, 108
- build\_id
  - btp\_core\_frame, 26
- callees
  - btp\_callgraph, 23
- callgraph.h, 49
  - btp\_callgraph\_extend, 50
- cluster.h, 51
  - btp\_cluster\_free, 51
  - btp\_cluster\_new, 51
  - btp\_dendrogram\_cut, 51
  - btp\_dendrogram\_free, 52
  - btp\_dendrogram\_new, 52
  - btp\_distances\_cluster\_objects, 52
- column
  - btp\_location, 42
- core\_fingerprint.h, 53
- core\_frame.h, 54
  - btp\_core\_frame\_append, 54
  - btp\_core\_frame\_append\_to\_str, 54
  - btp\_core\_frame\_cmp, 55



- btg\_gdb\_thread\_init, 83
  - btg\_gdb\_thread\_new, 83
  - btg\_gdb\_thread\_parse, 83
  - btg\_gdb\_thread\_parse\_funs, 83
  - btg\_gdb\_thread\_quality, 83
  - btg\_gdb\_thread\_quality\_counts, 84
  - btg\_gdb\_thread\_remove\_frame, 84
  - btg\_gdb\_thread\_remove\_frames\_above, 84
  - btg\_gdb\_thread\_remove\_frames\_below\_n, 84
  - btg\_gdb\_thread\_skip\_lwp, 84
- koops\_frame.h, 86
- btg\_koops\_frame\_dup, 86
  - btg\_koops\_frame\_free, 86
  - btg\_koops\_frame\_init, 87
  - btg\_koops\_frame\_new, 87
- koops\_stacktrace.h, 88
- btg\_koops\_stacktrace\_dup, 88
  - btg\_koops\_stacktrace\_free, 88
  - btg\_koops\_stacktrace\_init, 88
  - btg\_koops\_stacktrace\_new, 88
  - btg\_koops\_stacktrace\_parse, 89
- len
- btg\_strbuf, 46
- length
- btg\_elf\_frame\_description\_entry, 33
- library\_name
- btg\_gdb\_frame, 35
- libs
- btg\_gdb\_stacktrace, 38
- line
- btg\_location, 42
- location.h, 90
- btg\_location\_add, 90
  - btg\_location\_add\_ext, 91
  - btg\_location\_cmp, 91
  - btg\_location\_eat\_char, 91
  - btg\_location\_eat\_char\_ext, 91
  - btg\_location\_init, 91
  - btg\_location\_to\_string, 92
- merge\_levels
- btg\_dendrogram, 30
- message
- btg\_location, 42
- metrics.h, 93
- btg\_dist\_thread\_type, 94
  - btg\_distances\_dup, 94
  - btg\_distances\_free, 94
  - btg\_distances\_get\_distance, 94
  - btg\_distances\_new, 94
  - btg\_distances\_set\_distance, 95
  - btg\_gdb\_threads\_compare, 95
- next
- btg\_core\_frame, 26
  - btg\_core\_thread, 29
  - btg\_gdb\_frame, 35
  - btg\_gdb\_thread, 39
- normalize.h, 96
- btg\_glibc\_thread\_find\_exit\_frame, 96
  - btg\_normalize\_optimize\_thread, 96
  - btg\_normalize\_paired\_unknown\_function\_names, 96
- number
- btg\_gdb\_frame, 36
- python\_frame.h, 97
- python\_stacktrace.h, 98
- sha1.h, 99
- signal\_handler\_called
- btg\_gdb\_frame, 36
- source\_file
- btg\_gdb\_frame, 36
- source\_line
- btg\_gdb\_frame, 36
- start\_address
- btg\_elf\_frame\_description\_entry, 33
- strbuf.h, 100
- btg\_strbuf\_append\_char, 100
  - btg\_strbuf\_append\_str, 100
  - btg\_strbuf\_append\_strf, 100
  - btg\_strbuf\_append\_strfv, 101
  - btg\_strbuf\_clear, 101
  - btg\_strbuf\_free, 101
  - btg\_strbuf\_free\_nobuf, 101
  - btg\_strbuf\_grow, 101
  - btg\_strbuf\_init, 101
  - btg\_strbuf\_new, 101
  - btg\_strbuf\_prepend\_str, 101
  - btg\_strbuf\_prepend\_strf, 102
  - btg\_strbuf\_prepend\_strfv, 102
- symbol\_name
- btg\_elf\_plt\_entry, 34
- unstrip.h, 103
- utils.h, 104
- btg\_asprintf, 105
  - btg\_debug\_parser, 109
  - btg\_file\_to\_string, 105
  - btg\_malloc, 105
  - btg\_mallocz, 105
  - btg\_parse\_char\_cspan, 105
  - btg\_parse\_char\_limited, 105
  - btg\_parse\_char\_span, 106
  - btg\_parse\_digit, 106
  - btg\_parse\_hexadecimal\_number, 106

btp\_parse\_string, 106  
btp\_parse\_unsigned\_integer, 106  
btp\_realloc, 106  
btp\_skip\_char, 106  
btp\_skip\_char\_limited, 106  
btp\_skip\_char\_sequence, 107  
btp\_skip\_char\_span, 107  
btp\_skip\_char\_span\_location, 107  
btp\_skip\_hexadecimal\_number, 107  
btp\_skip\_string, 107  
btp\_skip\_unsigned\_integer, 107  
btp\_strchr\_location, 108  
btp\_strcmp0, 108  
btp\_strdup, 108  
btp\_strndup, 108  
btp\_strspn\_location, 108  
btp\_strstr\_location, 108  
btp\_vasprintf, 108