

Btparser

A program failure analysis library

Karel Klíč

November 26, 2012

Contents

1	Overview	7
I	Concepts	9
2	Stack Trace Normalization	11
3	Stack Trace Clustering	13
4	Core Dump Failure Analysis	15
5	Wishlist	17
II	Implementation	19
6	Overview	21
7	Data Structure Index	23
7.1	Data Structures	23
7.2	File List	24
8	Data Structure Documentation	27
8.1	btp_callgraph Struct Reference	27
8.1.1	Detailed Description	27
8.1.2	Field Documentation	28
8.2	btp_cluster Struct Reference	28
8.2.1	Detailed Description	28
8.3	btp_core_frame Struct Reference	28
8.3.1	Detailed Description	29
8.3.2	Field Documentation	29
8.4	btp_core_stacktrace Struct Reference	30
8.4.1	Detailed Description	30
8.4.2	Field Documentation	30
8.5	btp_core_thread Struct Reference	31
8.5.1	Detailed Description	31
8.5.2	Field Documentation	31
8.6	btp_deb_package Struct Reference	32
8.7	btp_dendrogram Struct Reference	32
8.7.1	Detailed Description	32
8.7.2	Field Documentation	32
8.8	btp_distances Struct Reference	33
8.8.1	Detailed Description	33
8.9	btp_elf_fde Struct Reference	33

8.9.1	Detailed Description	34
8.9.2	Field Documentation	34
8.10	btp_elf_plt_entry Struct Reference	34
8.10.1	Detailed Description	34
8.10.2	Field Documentation	35
8.11	btp_gdb_frame Struct Reference	35
8.11.1	Detailed Description	35
8.11.2	Field Documentation	36
8.12	btp_gdb_sharedlib Struct Reference	37
8.12.1	Detailed Description	37
8.13	btp_gdb_stacktrace Struct Reference	37
8.13.1	Detailed Description	38
8.13.2	Field Documentation	38
8.14	btp_gdb_thread Struct Reference	39
8.14.1	Detailed Description	39
8.14.2	Field Documentation	39
8.15	btp_java_exception Struct Reference	40
8.15.1	Detailed Description	40
8.15.2	Field Documentation	40
8.16	btp_java_frame Struct Reference	41
8.16.1	Field Documentation	41
8.17	btp_java_stacktrace Struct Reference	42
8.17.1	Field Documentation	42
8.18	btp_java_thread Struct Reference	43
8.18.1	Detailed Description	43
8.18.2	Field Documentation	43
8.19	btp_json_settings Struct Reference	44
8.20	btp_json_value Struct Reference	44
8.21	btp_koops_frame Struct Reference	45
8.21.1	Detailed Description	46
8.21.2	Field Documentation	46
8.22	btp_koops_stacktrace Struct Reference	47
8.22.1	Field Documentation	47
8.23	btp_location Struct Reference	48
8.23.1	Detailed Description	48
8.23.2	Field Documentation	48
8.24	btp_operating_system Struct Reference	49
8.25	btp_python_frame Struct Reference	49
8.26	btp_python_stacktrace Struct Reference	50
8.27	btp_report Struct Reference	50
8.28	btp_rpm_consistency Struct Reference	51
8.29	btp_rpm_package Struct Reference	52
8.30	btp_sha1_state Struct Reference	52
8.30.1	Detailed Description	53
8.31	btp_strbuf Struct Reference	53
8.31.1	Detailed Description	53
8.31.2	Field Documentation	53
8.32	btp_unstrip_entry Struct Reference	53
8.32.1	Detailed Description	54
9	File Documentation	55
9.1	callgraph.h File Reference	55
9.1.1	Detailed Description	56
9.1.2	Function Documentation	56

9.2	cluster.h File Reference	56
9.2.1	Detailed Description	56
9.2.2	Function Documentation	57
9.3	core_fingerprint.h File Reference	58
9.3.1	Detailed Description	58
9.4	core_frame.h File Reference	59
9.4.1	Detailed Description	59
9.4.2	Function Documentation	59
9.5	core_stacktrace.h File Reference	61
9.5.1	Detailed Description	62
9.5.2	Function Documentation	62
9.6	core_thread.h File Reference	63
9.6.1	Detailed Description	64
9.6.2	Function Documentation	64
9.7	deb.h File Reference	65
9.7.1	Detailed Description	66
9.8	disasm.h File Reference	66
9.8.1	Detailed Description	66
9.8.2	Function Documentation	67
9.9	elves.h File Reference	67
9.9.1	Detailed Description	68
9.9.2	Function Documentation	68
9.10	gdb_frame.h File Reference	68
9.10.1	Detailed Description	70
9.10.2	Function Documentation	70
9.11	gdb_sharedlib.h File Reference	76
9.11.1	Detailed Description	77
9.11.2	Function Documentation	77
9.12	gdb_stacktrace.h File Reference	78
9.12.1	Detailed Description	79
9.12.2	Function Documentation	79
9.13	gdb_thread.h File Reference	84
9.13.1	Detailed Description	85
9.13.2	Function Documentation	85
9.14	java_exception.h File Reference	88
9.14.1	Detailed Description	89
9.14.2	Function Documentation	90
9.15	java_frame.h File Reference	93
9.15.1	Detailed Description	94
9.15.2	Function Documentation	94
9.16	java_stacktrace.h File Reference	96
9.16.1	Detailed Description	96
9.16.2	Function Documentation	96
9.17	java_thread.h File Reference	98
9.17.1	Detailed Description	99
9.17.2	Function Documentation	99
9.18	koops_frame.h File Reference	102
9.18.1	Detailed Description	103
9.18.2	Function Documentation	103
9.19	koops_stacktrace.h File Reference	104
9.19.1	Detailed Description	105
9.19.2	Function Documentation	105
9.20	location.h File Reference	107
9.20.1	Detailed Description	107

9.20.2	Function Documentation	108
9.21	metrics.h File Reference	109
9.21.1	Detailed Description	111
9.21.2	Typedef Documentation	111
9.21.3	Function Documentation	111
9.22	normalize.h File Reference	112
9.22.1	Detailed Description	113
9.22.2	Function Documentation	113
9.23	python_frame.h File Reference	113
9.23.1	Detailed Description	114
9.23.2	Function Documentation	114
9.24	python_stacktrace.h File Reference	115
9.24.1	Detailed Description	116
9.24.2	Function Documentation	116
9.25	rpm.h File Reference	117
9.25.1	Detailed Description	117
9.25.2	Function Documentation	117
9.26	sha1.h File Reference	118
9.26.1	Detailed Description	118
9.27	strbuf.h File Reference	119
9.27.1	Detailed Description	119
9.27.2	Function Documentation	120
9.28	unstrip.h File Reference	121
9.28.1	Detailed Description	122
9.29	utils.h File Reference	122
9.29.1	Detailed Description	123
9.29.2	Function Documentation	123
9.29.3	Variable Documentation	128
10	Example Documentation	129
10.1	/home/karel/devel/btparser/lib/koops_frame.h	129
11	Known Bugs	131
12	Wishlist	133
	Index	134

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	28
btp_core_frame	A function call on call stack of a core dump	28
btp_core_stacktrace	A stack trace of a core dump	30
btp_core_thread	A thread of execution on call stack of a core dump	31
btp_deb_package	32
btp_dendrogram	A dendrogram created by clustering	32
btp_distances	A distance matrix of stack trace threads	33
btp_elf_fde	A single Frame Description Entry of the .eh_frame section present in ELF binaries . . .	33
btp_elf_plt_entry	A single item of the Procedure Linkage Table present in ELF binaries	34
btp_gdb_frame	A function call of a GDB-produced stack trace	35
btp_gdb_sharedlib	A shared library memory location as reported by GDB	37
btp_gdb_stacktrace	A stack trace produced by GDB	37
btp_gdb_thread	A thread of execution of a GDB-produced stack trace	39
btp_java_exception	A exception of execution of a JAVA-produced stack trace	40
btp_java_frame	41
btp_java_stacktrace	42
btp_java_thread	A thread of execution of a JAVA-produced stack trace	43

btp_json_settings	44
btp_json_value	44
btp_koops_frame	
Kernel oops stack frame	45
btp_koops_stacktrace	47
btp_location	
A location of a parser in the input stream	48
btp_operating_system	49
btp_python_frame	49
btp_python_stacktrace	50
btp_report	50
btp_rpm_consistency	51
btp_rpm_package	52
btp_sha1_state	
Internal state of a SHA-1 hash algorithm	52
btp_strbuf	
A resizable string buffer	53
btp_unstrip_entry	
Core dump memory layout as reported by the unstrip utility	53

7.2 File List

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

callgraph.h	
Calling relationships between subroutines	55
cluster.h	
Clustering for stack trace threads	56
config.h	??
core_fingerprint.h	
Fingerprint algorithm for core stack traces	58
core_frame.h	
Single frame of core stack trace thread	59
core_stacktrace.h	
A stack trace of a core dump	61
core_thread.h	
Single thread of execution of a core stack trace	63
core_unwind.h	??
deb.h	
Deb-related structures and utilities	65
disasm.h	
BFD-based function disassembler	66
elves.h	
Loading PLT and FDEs from ELF binaries	67
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	78
gdb_thread.h	
Single thread of execution of GDB stack trace	84

java_exception.h	Single exception of execution of JAVA stack trace	88
java_frame.h	Java frame structure and related algorithms	93
java_stacktrace.h	Java stack trace structure and related algorithms	96
java_thread.h	Single thread of execution of JAVA stack trace	98
json.h		??
koops_frame.h	Kernel oops stack frame	102
koops_stacktrace.h	Kernel oops stack trace structure and related algorithms	104
location.h	Parser location in input file	107
metrics.h	Distance between stack trace threads	109
normalize.h	Normalization of stack traces	112
python_frame.h	Python frame structure and related algorithms	113
python_stacktrace.h	Python stack trace structure and related algorithms	115
report.h		??
rpm.h	RPM-related structures and utilities	117
sha1.h	An implementation of SHA-1 cryptographic hash function	118
strbuf.h	A string buffer structure and related algorithms	119
unstrip.h	Parser for the output of the unstrip utility	121
utils.h	Various utility functions, macros and variables that do not fit elsewhere	122

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:

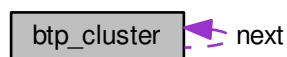
- `callgraph.h`

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:

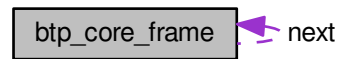
- `cluster.h`

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`

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

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

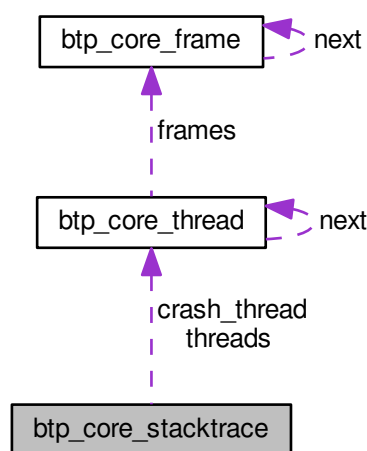
- core_frame.h

8.4 btp_core_stacktrace Struct Reference

A stack trace of a core dump.

```
#include <core_stacktrace.h>
```

Collaboration diagram for btp_core_stacktrace:



Data Fields

- `uint8_t` `signal`
- `char *` **`executable`**
- `struct btp_core_thread *` `crash_thread`
Thread responsible for the crash.
- `struct btp_core_thread *` **`threads`**

8.4.1 Detailed Description

A stack trace of a core dump.

8.4.2 Field Documentation

8.4.2.1 `struct btp_core_thread* btp_core_stacktrace::crash_thread`

Thread responsible for the crash.

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

8.4.2.2 `uint8_t btp_core_stacktrace::signal`

Signal number.

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

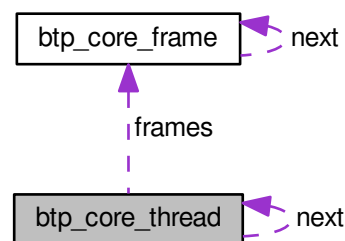
- `core_stacktrace.h`

8.5 `btp_core_thread` Struct Reference

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

```
#include <core_thread.h>
```

Collaboration diagram for `btp_core_thread`:



Data Fields

- `struct btp_core_frame * frames`
- `struct btp_core_thread * next`

8.5.1 Detailed Description

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

8.5.2 Field Documentation

8.5.2.1 `struct btp_core_frame* btp_core_thread::frames`

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

8.5.2.2 `struct btp_core_thread* btp_core_thread::next`

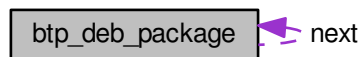
A sibling thread, or NULL if this is the last thread in a stacktrace.

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

- `core_thread.h`

8.6 `btp_deb_package` Struct Reference

Collaboration diagram for `btp_deb_package`:



Data Fields

- `struct btp_deb_package * next`

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

- `deb.h`

8.7 `btp_dendrogram` Struct Reference

A dendrogram created by clustering.

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

Data Fields

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

8.7.1 Detailed Description

A dendrogram created by clustering.

8.7.2 Field Documentation

8.7.2.1 `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`

8.8 `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:

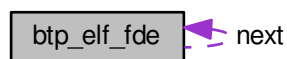
- `metrics.h`

8.9 `btp_elf_fde` Struct Reference

A single Frame Description Entry of the `.eh_frame` section present in ELF binaries.

```
#include <elves.h>
```

Collaboration diagram for `btp_elf_fde`:



Data Fields

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

8.9.1 Detailed Description

A single Frame Description Entry of the `.eh_frame` section present in ELF binaries.

8.9.2 Field Documentation

8.9.2.1 `uint64_t btp_elf_fde::length`

Length of the function in bytes.

8.9.2.2 `uint64_t btp_elf_fde::start_address`

Offset where a function starts. If the function is present in the Procedure Linkage Table, this address matches some address in `btp_elf_plt_entry`.

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

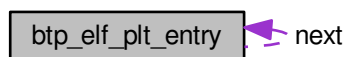
- `elves.h`

8.10 `btp_elf_plt_entry` Struct Reference

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

`#include <elves.h>`

Collaboration diagram for `btp_elf_plt_entry`:



Data Fields

- `uint64_t address`
- `char * symbol_name`
- `struct btp_elf_plt_entry * next`

8.10.1 Detailed Description

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

8.10.2 Field Documentation

8.10.2.1 `uint64_t btp_elf_plt_entry::address`

Address of the entry.

8.10.2.2 `char* btp_elf_plt_entry::symbol_name`

Symbol name corresponding to the address.

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

- `elves.h`

8.11 `btp_gdb_frame` Struct Reference

A function call of a GDB-produced stack trace.

```
#include <gdb_frame.h>
```

Collaboration diagram for `btp_gdb_frame`:



Data Fields

- `char * function_name`
- `char * function_type`
- `uint32_t number`
- `char * source_file`
- `uint32_t source_line`
- `bool signal_handler_called`
- `uint64_t address`
- `char * library_name`
- `struct btp_gdb_frame * next`

8.11.1 Detailed Description

A function call of a GDB-produced stack trace.

A frame representing a function call or a signal handler on a call stack of a thread.

8.11.2 Field Documentation

8.11.2.1 `uint64_t btp_gdb_frame::address`

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

8.11.2.2 `char* btp_gdb_frame::function_name`

A function name or NULL. If it's NULL, `signal_handler_called` is true.

8.11.2.3 `char* btp_gdb_frame::function_type`

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

8.11.2.4 `char* btp_gdb_frame::library_name`

A library name or NULL.

8.11.2.5 `struct btp_gdb_frame* btp_gdb_frame::next`

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

8.11.2.6 `uint32_t btp_gdb_frame::number`

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

8.11.2.7 `bool btp_gdb_frame::signal_handler_called`

Signal handler was called on this frame.

8.11.2.8 `char* btp_gdb_frame::source_file`

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

8.11.2.9 `uint32_t btp_gdb_frame::source_line`

A line number in the source file, determining the position of the function definition, or -1 when unknown.

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

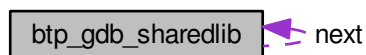
- `gdb_frame.h`

8.12 btp_gdb_sharedlib Struct Reference

A shared library memory location as reported by GDB.

```
#include <gdb_sharedlib.h>
```

Collaboration diagram for btp_gdb_sharedlib:



Data Fields

- uint64_t **from**
- uint64_t **to**
- int **symbols**
- char * **soname**
- struct btp_gdb_sharedlib * **next**

8.12.1 Detailed Description

A shared library memory location as reported by GDB.

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

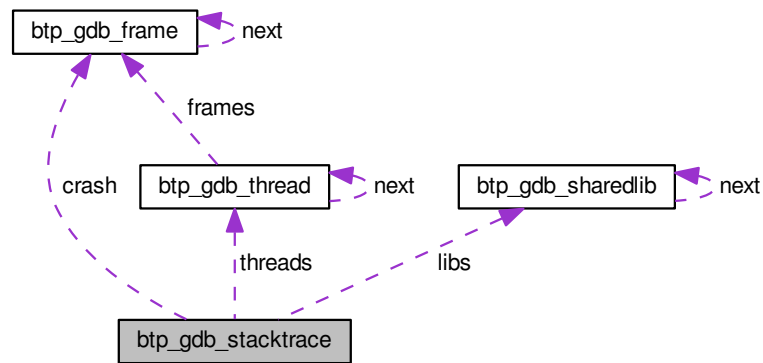
- gdb_sharedlib.h

8.13 btp_gdb_stacktrace Struct Reference

A stack trace produced by GDB.

```
#include <gdb_stacktrace.h>
```

Collaboration diagram for `btp_gdb_stacktrace`:



Data Fields

- `struct btp_gdb_thread * threads`
- `struct btp_gdb_frame * crash`
- `struct btp_gdb_sharedlib * libs`

8.13.1 Detailed Description

A stack trace produced by GDB.

A stacktrace obtained at the time of a program crash, consisting of several threads which contains frames.

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

8.13.2 Field Documentation

8.13.2.1 `struct btp_gdb_frame* btp_gdb_stacktrace::crash`

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`

Shared libraries loaded at the moment of crash.

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

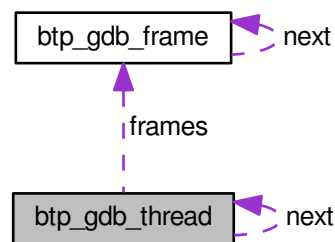
- `gdb_stacktrace.h`

8.14 `btp_gdb_thread` Struct Reference

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

```
#include <gdb_thread.h>
```

Collaboration diagram for `btp_gdb_thread`:



Data Fields

- `uint32_t number`
- `struct btp_gdb_frame * frames`
- `struct btp_gdb_thread * next`

8.14.1 Detailed Description

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

Represents a thread containing frames.

8.14.2 Field Documentation

8.14.2.1 `struct btp_gdb_frame* btp_gdb_thread::frames`

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

8.14.2.2 `struct btp_gdb_thread* btp_gdb_thread::next`

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:

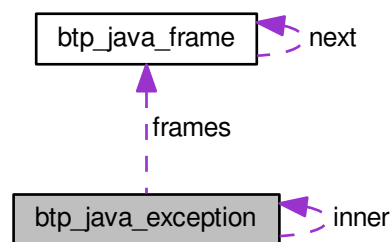
- `gdb_thread.h`

8.15 `btp_java_exception` Struct Reference

A exception of execution of a JAVA-produced stack trace.

```
#include <java_exception.h>
```

Collaboration diagram for `btp_java_exception`:



Data Fields

- `char * name`
- `char * message`
- `struct btp_java_frame * frames`
- `struct btp_java_exception * inner`

8.15.1 Detailed Description

A exception of execution of a JAVA-produced stack trace.

Represents a exception containing frames.

8.15.2 Field Documentation

8.15.2.1 `struct btp_java_frame* btp_java_exception::frames`

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

8.15.2.2 `struct btp_java_exception* btp_java_exception::inner`

An inner exception, or NULL if this exception doesn't have an inner exception

8.15.2.3 `char* btp_java_exception::message`

Message delivered by the exception. Can be NULL

8.15.2.4 `char* btp_java_exception::name`

Exception caught in this exception. Can be NULL

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

- `java_exception.h`

8.16 `btp_java_frame` Struct Reference

Collaboration diagram for `btp_java_frame`:



Data Fields

- `char * file_name`
- `uint32_t file_line`
- `char * class_path`
- `char * function_name`
- `bool is_native`
- `struct btp_java_frame * next`

8.16.1 Field Documentation

8.16.1.1 `char* btp_java_frame::class_path`

A path to jar file or class file. Can be NULL

8.16.1.2 `uint32_t btp_java_frame::file_line`

Line no. in the Java file. 0 is used when `file_line` is missing.

8.16.1.3 `char* btp_java_frame::file_name`

a Java file. Can be NULL

8.16.1.4 `char* btp_java_frame::function_name`

FQDN - Fully qualified domain name. Can be NULL `<Namespace>.<Type>.<Function name>=""`

8.16.1.5 bool `btp_java_frame::is_native`

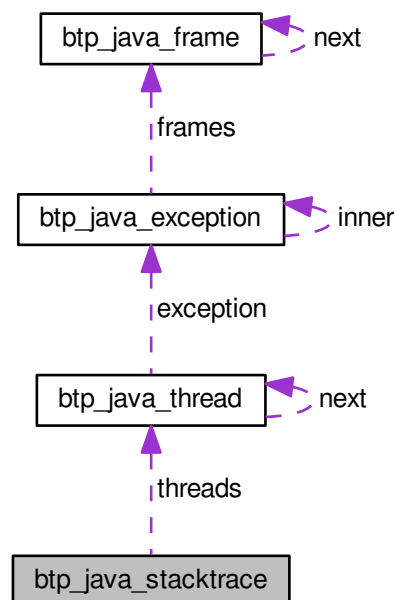
True if method is native.

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

- `java_frame.h`

8.17 `btp_java_stacktrace` Struct Reference

Collaboration diagram for `btp_java_stacktrace`:



Data Fields

- `struct btp_java_thread * threads`

8.17.1 Field Documentation

8.17.1.1 `struct btp_java_thread* btp_java_stacktrace::threads`

Threads of stack trace. Always non-NULL.

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

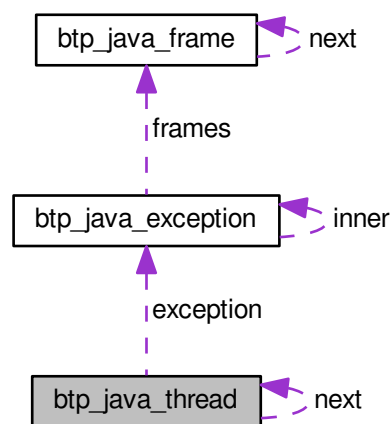
- `java_stacktrace.h`

8.18 `btp_java_thread` Struct Reference

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

`#include <java_thread.h>`

Collaboration diagram for `btp_java_thread`:



Data Fields

- `char * name`
- `struct btp_java_exception * exception`
- `struct btp_java_thread * next`

8.18.1 Detailed Description

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

Represents a thread containing frames.

8.18.2 Field Documentation

8.18.2.1 `struct btp_java_exception* btp_java_thread::exception`

Thread's exception. Can be NULL

8.18.2.2 `char* btp_java_thread::name`

Thread name. Can be NULL

8.18.2.3 `struct btp_java_thread* btp_java_thread::next`

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:

- `java_thread.h`

8.19 `btp_json_settings` Struct Reference

Data Fields

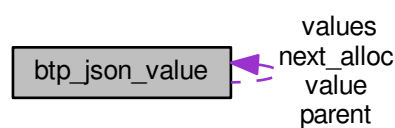
- unsigned long **max_memory**
- int **settings**

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

- `json.h`

8.20 `btp_json_value` Struct Reference

Collaboration diagram for `btp_json_value`:



Data Fields

- struct `btp_json_value *` **parent**
- enum `btp_json_type` **type**
- union {
 - int **boolean**
 - long **integer**
 - double **dbl**
 - struct {

```

    unsigned length
    char * ptr
} string
struct {
    unsigned length
    struct {
        char * name
        struct btp_json_value * value
    } values
} object
struct {
    unsigned length
    struct btp_json_value ** values
} array
} u

• union {
    struct btp_json_value * next_alloc
    void * object_mem
} _reserved

```

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

- json.h

8.21 btp_koops_frame Struct Reference

Kernel oops stack frame.

```
#include <koops_frame.h>
```

Collaboration diagram for btp_koops_frame:



Data Fields

- uint64_t **address**
- bool **reliable**
- char * **function_name**
- uint64_t **function_offset**
- uint64_t **function_length**

- `char * module_name`
- `uint64_t from_address`
- `char * from_function_name`
- `uint64_t from_function_offset`
- `uint64_t from_function_length`
- `char * from_module_name`
- `struct btp_koops_frame * next`

8.21.1 Detailed Description

Kernel oops stack frame.

8.21.2 Field Documentation

8.21.2.1 `uint64_t btp_koops_frame::address`

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

8.21.2.2 `uint64_t btp_koops_frame::from_address`

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

8.21.2.3 `char* btp_koops_frame::from_function_name`

Might be NULL.

8.21.2.4 `char* btp_koops_frame::from_module_name`

Might be NULL.

8.21.2.5 `char* btp_koops_frame::function_name`

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

8.21.2.6 `char* btp_koops_frame::module_name`

Might be NULL.

8.21.2.7 `bool btp_koops_frame::reliable`

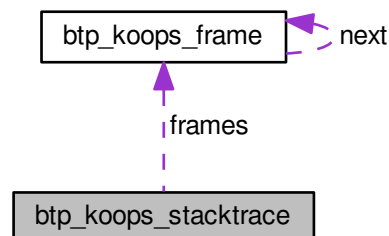
<http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git;a=blob;f=arch/x86/kernel/dumpstack.c>
`printk_address(unsigned long address, int reliable)`

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

- `koops_frame.h`

8.22 btp_koops_stacktrace Struct Reference

Collaboration diagram for btp_koops_stacktrace:



Data Fields

- char * version
Version of the kernel.
- bool taint_module_proprietary
- bool **taint_module_gpl**
- bool **taint_module_out_of_tree**
- bool **taint_forced_module**
- bool **taint_forced_removal**
- bool **taint_smp_unsafe**
- bool taint_mce
- bool taint_page_release
- bool **taint_userspace**
- bool **taint_died_recently**
- bool **taint_acpi_overridden**
- bool **taint_warning**
- bool **taint_staging_driver**
- bool **taint_firmware_workaround**
- bool **taint_virtual_box**
- char ** modules
List of loaded modules.
- struct btp_koops_frame * frames
Call trace. It might be NULL as it is not mandatory.

8.22.1 Field Documentation

8.22.1.1 char** btp_koops_stacktrace::modules

List of loaded modules.

It might be NULL as it is sometimes not included in a kerneloops.

8.22.1.2 `bool btp_koops_stacktrace::taint_mce`

A machine check exception has been raised.

8.22.1.3 `bool btp_koops_stacktrace::taint_module_proprietary`

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

8.22.1.4 `bool btp_koops_stacktrace::taint_page_release`

A process has been found in a bad page state.

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

- `koops_stacktrace.h`

8.23 `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`

8.23.1 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.

8.23.2 Field Documentation

8.23.2.1 `int btp_location::column`

Starts from 0.

8.23.2.2 `int btp_location::line`

Starts from 1.

8.23.2.3 `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`

8.24 `btp_operating_system` Struct Reference

Data Fields

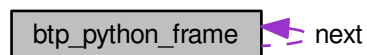
- `char * name`
- `char * version`
- `char * architecture`
- `uint64_t uptime`

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

- `report.h`

8.25 `btp_python_frame` Struct Reference

Collaboration diagram for `btp_python_frame`:



Data Fields

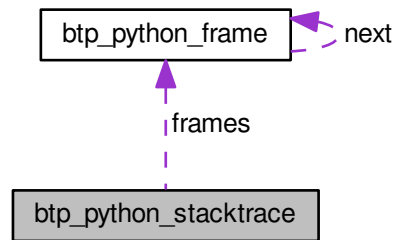
- `char * file_name`
- `uint32_t file_line`
- `bool is_module`
- `char * function_name`
- `char * line`
- `struct btp_python_frame * next`

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

- `python_frame.h`

8.26 btp_python_stacktrace Struct Reference

Collaboration diagram for btp_python_stacktrace:



Data Fields

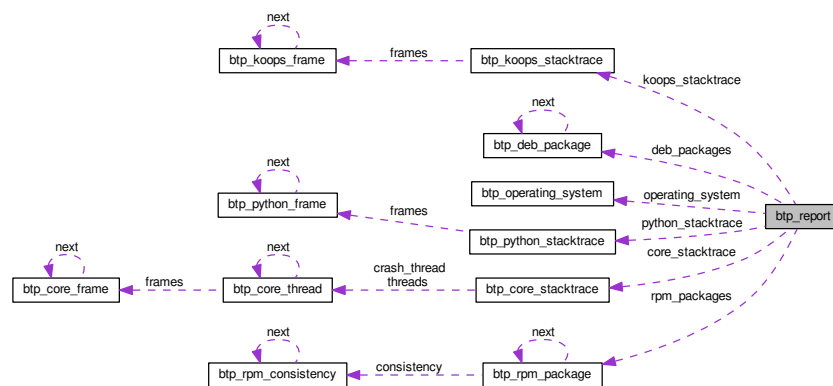
- char * **file_name**
- uint32_t **file_line**
- char * **exception_name**
- struct btp_python_frame * **frames**

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

- python_stacktrace.h

8.27 btp_report Struct Reference

Collaboration diagram for btp_report:



Data Fields

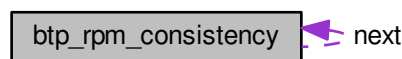
- uint32_t **report_version**
- enum btp_report_type **report_type**
- char * **reporter_name**
- char * **reporter_version**
- enum btp_user_type **user_type**
- struct btp_operating_system **operating_system**
- char * **component_name**
- struct btp_rpm_package * **rpm_packages**
- struct btp_deb_package * **deb_packages**
- struct btp_python_stacktrace * **python_stacktrace**
- struct btp_koops_stacktrace * **koops_stacktrace**
- struct btp_core_stacktrace * **core_stacktrace**

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

- report.h

8.28 btp_rpm_consistency Struct Reference

Collaboration diagram for btp_rpm_consistency:



Data Fields

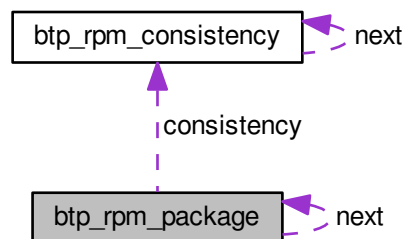
- char * **file_name**
- bool **owner_changed**
- bool **group_changed**
- bool **mode_changed**
- bool **md5_mismatch**
- bool **size_changed**
- bool **major_number_changed**
- bool **minor_number_changed**
- bool **symlink_changed**
- bool **modification_time_changed**
- struct btp_rpm_consistency * **next**

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

- rpm.h

8.29 btp_rpm_package Struct Reference

Collaboration diagram for btp_rpm_package:



Data Fields

- `char * name`
- `uint32_t epoch`
- `char * version`
- `char * release`
- `char * architecture`
- `uint32_t install_time`
- `struct btp_rpm_consistency * consistency`
- `struct btp_rpm_package * next`

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

- `rpm.h`

8.30 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]`

8.30.1 Detailed Description

Internal state of a SHA-1 hash algorithm.

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

- `sha1.h`

8.31 **btp_strbuf** Struct Reference

A resizable string buffer.

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

Data Fields

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

8.31.1 Detailed Description

A resizable string buffer.

8.31.2 Field Documentation

8.31.2.1 `int btp_strbuf::alloc`

Size of the allocated buffer. Always > 0.

8.31.2.2 `int btp_strbuf::len`

Length of the string, without the ending `\0`.

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

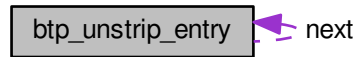
- `strbuf.h`

8.32 **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**

8.32.1 Detailed Description

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

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

- `unstrip.h`

Chapter 9

File Documentation

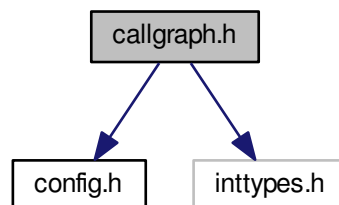
9.1 callgraph.h File Reference

Calling relationships between subroutines.

```
#include "config.h"
```

```
#include <inttypes.h>
```

Include dependency graph for callgraph.h:



Data Structures

- `struct btp_callgraph`

A call graph representing calling relationships between subroutines.

Functions

- `struct btp_callgraph * btp_callgraph_compute (struct btp_disasm_state *disassembler, struct btp_elf_fde *eh_frame, char **error_message)`
- `struct btp_callgraph * btp_callgraph_extend (struct btp_callgraph *callgraph, uint64_t start_address, struct btp_disasm_state *disassembler, struct btp_elf_fde *eh_frame, char **error_message)`
- `void btp_callgraph_free (struct btp_callgraph *callgraph)`

- struct btp_callgraph * **btp_callgraph_find** (struct btp_callgraph *callgraph, uint64_t address)
- struct btp_callgraph * **btp_callgraph_last** (struct btp_callgraph *callgraph)

9.1.1 Detailed Description

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

Call graph is used by fingerprinting algorithms.

9.1.2 Function Documentation

- 9.1.2.1 struct btp_callgraph* btp_callgraph_extend (struct btp_callgraph * *callgraph*, uint64_t *start_address*, struct btp_disasm_state * *disassembler*, struct btp_elf_fde * *eh_frame*, char ** *error_message*) [read]

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

9.2 cluster.h File Reference

Clustering for stack trace threads.

Data Structures

- struct btp_dendrogram
A dendrogram created by clustering.
- struct btp_cluster
A cluster of objects from a dendrogram.

Functions

- struct btp_dendrogram * btp_dendrogram_new (int size)
- void btp_dendrogram_free (struct btp_dendrogram *dendrogram)
- struct btp_dendrogram * btp_distances_cluster_objects (struct btp_distances *distances)
- struct btp_cluster * btp_cluster_new (int size)
- void btp_cluster_free (struct btp_cluster *cluster)
- struct btp_cluster * btp_dendrogram_cut (struct btp_dendrogram *dendrogram, float level, int min_size)

9.2.1 Detailed Description

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

9.2.2 Function Documentation

9.2.2.1 void `btp_cluster_free` (struct `btp_cluster` * *cluster*)

Releases the memory held by the cluster.

Parameters

<i>dendrogram</i>	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

<i>size</i>	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

<i>dendrogram</i>	The dendrogram which should be cut. The structure is not modified by this call.
<i>level</i>	The cutting level of distance.
<i>min_size</i>	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

<i>dendrogram</i>	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

<i>size</i>	Number of objects.
-------------	--------------------

Returns

It never returns NULL. The returned pointer must be released by `btb_dendrogram_free()`.

9.2.2.6 `struct btb_dendrogram* btb_distances_cluster_objects (struct btb_distances * distances)` [read]

Performs hierarchical agglomerative clustering on objects.

Parameters

<i>distances</i>	Distances between the objects. The structure is not modified by calling this function.
------------------	--

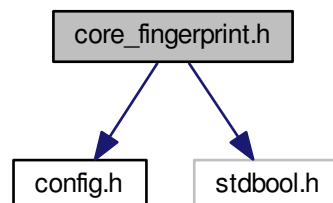
9.3 `core_fingerprint.h` File Reference

Fingerprint algorithm for core stack traces.

```
#include "config.h"
```

```
#include <stdbool.h>
```

Include dependency graph for `core_fingerprint.h`:

**Functions**

- `bool btb_core_fingerprint_generate (struct btb_core_stacktrace *stacktrace, char **error_message)`
- `bool btb_core_fingerprint_generate_for_binary (struct btb_core_thread *thread, const char *binary_path, char **error_message)`

9.3.1 Detailed Description

Fingerprint algorithm for core stack traces.

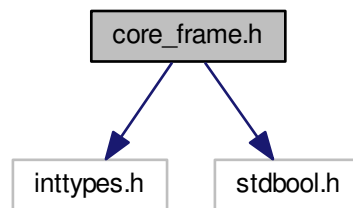
9.4 core_frame.h File Reference

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)
- char * btp_core_frame_to_json (struct btp_core_frame *frame)

9.4.1 Detailed Description

Single frame of core stack trace thread.

9.4.2 Function Documentation

9.4.2.1 struct btp_core_frame* btp_core_frame_append (struct btp_core_frame * *dest*, struct btp_core_frame * *item*) [read]

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

Returns

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

9.4.2.2 int `btp_core_frame_cmp` (struct `btp_core_frame` * *frame1*, struct `btp_core_frame` * *frame2*)

Compares two frames.

Parameters

<i>frame1</i>	It must be non-NULL pointer. It's not modified by calling this function.
<i>frame2</i>	It must be non-NULL pointer. It's not modified by calling this function.

Returns

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

**9.4.2.3 struct `btp_core_frame`* `btp_core_frame_dup` (struct `btp_core_frame` * *frame*, bool *siblings*)
[read]**

Creates a duplicate of the frame.

Parameters

<i>frame</i>	It must be non-NULL pointer. The frame is not modified by calling this function.
<i>siblings</i>	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()`.

9.4.2.4 void `btp_core_frame_free` (struct `btp_core_frame` * *frame*)

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

Parameters

<i>frame</i>	If the frame is NULL, no operation is performed.
--------------	--

9.4.2.5 void `btp_core_frame_init` (struct `btp_core_frame` * *frame*)

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

9.4.2.6 struct btp_core_frame* btp_core_frame_new () [read]

Creates and initializes a new frame structure.

Returns

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

9.4.2.7 char* btp_core_frame_to_json (struct btp_core_frame * frame)

Returns a textual representation of the frame.

Parameters

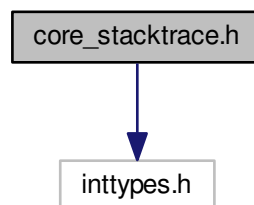
<i>frame</i>	It must be a non-NULL pointer. It's not modified by calling this function.
--------------	--

9.5 core_stacktrace.h File Reference

A stack trace of a core dump.

#include <inttypes.h>

Include dependency graph for core_stacktrace.h:



Data Structures

- struct btp_core_stacktrace
A stack trace of a core dump.

Functions

- struct btp_core_stacktrace * btp_core_stacktrace_new ()
- void btp_core_stacktrace_init (struct btp_core_stacktrace *stacktrace)
- void btp_core_stacktrace_free (struct btp_core_stacktrace *stacktrace)
- struct btp_core_stacktrace * btp_core_stacktrace_dup (struct btp_core_stacktrace *stacktrace)

- `int btp_core_stacktrace_get_thread_count (struct btp_core_stacktrace *stacktrace)`
- `struct btp_core_stacktrace * btp_core_stacktrace_parse (const char **input, struct btp_location *location)`
- `char * btp_core_stacktrace_to_json (struct btp_core_stacktrace *stacktrace)`
- `struct btp_core_stacktrace * btp_core_stacktrace_create (const char *gdb_stacktrace_text, const char *unstrip_text, const char *executable_path)`

9.5.1 Detailed Description

A stack trace of a core dump.

9.5.2 Function Documentation

9.5.2.1 `struct btp_core_stacktrace* btp_core_stacktrace_dup (struct btp_core_stacktrace * stacktrace) [read]`

Creates a duplicate of the stacktrace.

Parameters

<i>stacktrace</i>	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 btp_core_stacktrace_free (struct btp_core_stacktrace * stacktrace)`

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

Parameters

<i>stacktrace</i>	If the stacktrace is NULL, no operation is performed.
-------------------	---

9.5.2.3 `int btp_core_stacktrace_get_thread_count (struct btp_core_stacktrace * stacktrace)`

Returns a number of threads in the stacktrace.

Parameters

<i>stacktrace</i>	It's not modified by calling this function.
-------------------	---

9.5.2.4 `void btp_core_stacktrace_init (struct btp_core_stacktrace * stacktrace)`

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

9.5.2.5 struct btp_core_stacktrace* btp_core_stacktrace_new () [read]

Creates and initializes a new stacktrace structure.

Returns

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

9.5.2.6 struct btp_core_stacktrace* btp_core_stacktrace_parse (const char ** input, struct btp_location * location) [read]

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

Note

Stacktrace can be serialized to string via btp_core_stacktrace_to_text().

9.5.2.7 char* btp_core_stacktrace_to_json (struct btp_core_stacktrace * stacktrace)

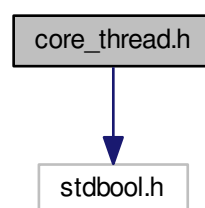
Serializes stacktrace to string. Newly allocated memory containing the textual representation of the provided stacktrace. Caller should free the memory when it's no longer needed.

9.6 core_thread.h File Reference

Single thread of execution of a core stack trace.

#include <stdbool.h>

Include dependency graph for core_thread.h:



Data Structures

- struct btp_core_thread

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

Functions

- `struct btp_core_thread * btp_core_thread_new ()`
- `void btp_core_thread_init (struct btp_core_thread *thread)`
- `void btp_core_thread_free (struct btp_core_thread *thread)`
- `struct btp_core_thread * btp_core_thread_dup (struct btp_core_thread *thread, bool siblings)`
- `int btp_core_thread_cmp (struct btp_core_thread *thread1, struct btp_core_thread *thread2)`
- `struct btp_core_thread * btp_core_thread_append (struct btp_core_thread *dest, struct btp_core_thread *item)`
- `int btp_core_thread_get_frame_count (struct btp_core_thread *thread)`
- `char * btp_core_thread_to_json (struct btp_core_thread *thread)`

9.6.1 Detailed Description

Single thread of execution of a core stack trace.

9.6.2 Function Documentation

9.6.2.1 `struct btp_core_thread* btp_core_thread_append (struct btp_core_thread * dest, struct btp_core_thread * item)` [read]

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

Returns

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

9.6.2.2 `int btp_core_thread_cmp (struct btp_core_thread * thread1, struct btp_core_thread * thread2)`

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

Returns

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

9.6.2.3 `struct btp_core_thread* btp_core_thread_dup (struct btp_core_thread * thread, bool siblings)` [read]

Creates a duplicate of the thread.

Parameters

<i>thread</i>	It must be non-NULL pointer. The thread is not modified by calling this function.
<i>siblings</i>	Whether to duplicate also siblings referenced by thread->next. If false, thread->next is not duplicated for the new frame, but it is set to NULL.

9.6.2.4 void `btp_core_thread_free` (struct `btp_core_thread` * *thread*)

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

Parameters

<i>thread</i>	If thread is NULL, no operation is performed.
---------------	---

9.6.2.5 int `btp_core_thread_get_frame_count` (struct `btp_core_thread` * *thread*)

Returns the number of frames in the thread.

9.6.2.6 void `btp_core_thread_init` (struct `btp_core_thread` * *thread*)

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

9.6.2.7 struct `btp_core_thread`* `btp_core_thread_new` () [read]

Creates and initializes a new frame structure.

Returns

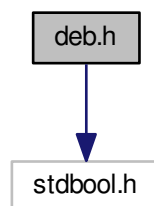
It never returns NULL. The returned pointer must be released by calling the function `btp_core_thread_free()`.

9.7 deb.h File Reference

Deb-related structures and utilities.

```
#include <stdbool.h>
```

Include dependency graph for deb.h:



Data Structures

- struct `btp_deb_package`

Functions

- void **btp_deb_package_free** (struct btp_deb_package *package, bool recursive)

9.7.1 Detailed Description

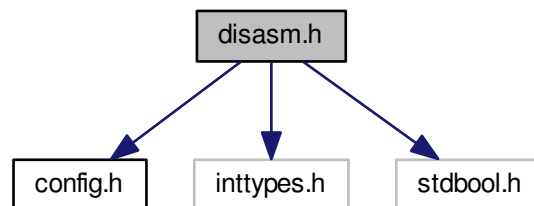
Deb-related structures and utilities.

9.8 disasm.h File Reference

BFD-based function disassembler.

```
#include "config.h"
#include <inttypes.h>
#include <stdbool.h>
```

Include dependency graph for disasm.h:



Functions

- struct btp_disasm_state * **btp_disasm_init** (const char *file_name, char **error_message)
- void **btp_disasm_free** (struct btp_disasm_state *state)
- char ** btp_disasm_get_function_instructions (struct btp_disasm_state *state, uint64_t start_offset, uint64_t size, char **error_message)
- void **btp_disasm_instructions_free** (char **instructions)
- bool **btp_disasm_instruction_is_one_of** (char *instruction, const char **mnemonics)
- bool **btp_disasm_instruction_present** (char **instructions, const char **mnemonics)
- bool **btp_disasm_instruction_parse_single_address_operand** (char *instruction, uint64_t *dest)
- uint64_t * **btp_disasm_get_callee_addresses** (char **instructions)

9.8.1 Detailed Description

BFD-based function disassembler.

9.8.2 Function Documentation

9.8.2.1 `char** btp_disasm_get_function_instructions (struct btp_disasm_state * state, uint64_t start_offset, uint64_t size, char ** error_message)`

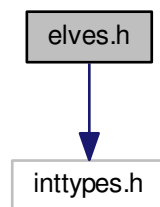
Disassemble the function starting at 'start_offset' and taking 'size' bytes, returning a list of (char*) instructions.

9.9 elves.h File Reference

Loading PLT and FDEs from ELF binaries.

`#include <inttypes.h>`

Include dependency graph for `elves.h`:



Data Structures

- `struct btp_elf_plt_entry`
A single item of the Procedure Linkage Table present in ELF binaries.
- `struct btp_elf_fde`
A single Frame Description Entry of the .eh_frame section present in ELF binaries.

Functions

- `struct btp_elf_plt_entry * btp_elf_get_procedure_linkage_table (const char *filename, char **error_message)`
- `void btp_elf_procedure_linkage_table_free (struct btp_elf_plt_entry *entries)`
- `struct btp_elf_plt_entry * btp_elf_plt_find_for_address (struct btp_elf_plt_entry *plt, uint64_t address)`
- `struct btp_elf_fde * btp_elf_get_eh_frame (const char *filename, char **error_message)`
- `void btp_elf_eh_frame_free (struct btp_elf_fde *entries)`
- `struct btp_elf_fde * btp_elf_find_fde_for_address (struct btp_elf_fde *eh_frame, uint64_t build_id_offset)`

9.9.1 Detailed Description

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

9.9.2 Function Documentation

9.9.2.1 `struct btp_elf_fde* btp_elf_get_elf_frame (const char * filename, char ** error_message)`
[read]

Reads the .eh_frame section from an ELF file.

Parameters

<i>error_message</i>	Will be filled by an error message if the function fails (returns NULL). Caller is responsible for calling free() on the string pointer. If function succeeds, the pointer is not touched by the function.
----------------------	--

Returns

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

9.9.2.2 `struct btp_elf_plt_entry* btp_elf_get_procedure_linkage_table (const char * filename, char ** error_message)` [read]

Reads the Procedure Linkage Table from an ELF file.

Parameters

<i>error_message</i>	Will be filled by an error message if the function fails (returns NULL). Caller is responsible for calling free() on the string pointer. If function succeeds, the pointer is not touched by the function.
----------------------	--

Returns

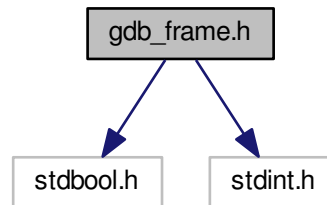
Linked list of PLT entries on success. NULL otherwise.

9.10 gdb_frame.h File Reference

Single frame of GDB stack trace thread.

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

Include dependency graph for gdb_frame.h:



Data Structures

- struct btp_gdb_frame
A function call of a GDB-produced stack trace.

Functions

- struct btp_gdb_frame * btp_gdb_frame_new ()
- void btp_gdb_frame_init (struct btp_gdb_frame *frame)
- void btp_gdb_frame_free (struct btp_gdb_frame *frame)
- struct btp_gdb_frame * btp_gdb_frame_dup (struct btp_gdb_frame *frame, bool siblings)
- bool btp_gdb_frame_calls_func (struct btp_gdb_frame *frame, const char *function_name,...)
- int btp_gdb_frame_cmp (struct btp_gdb_frame *frame1, struct btp_gdb_frame *frame2, bool compare_number)
- int btp_gdb_frame_cmp_simple (struct btp_gdb_frame *frame1, struct btp_gdb_frame *frame2)
- struct btp_gdb_frame * btp_gdb_frame_append (struct btp_gdb_frame *dest, struct btp_gdb_frame *item)
- void btp_gdb_frame_append_to_str (struct btp_gdb_frame *frame, struct btp_strbuf *dest, bool verbose)
- struct btp_gdb_frame * btp_gdb_frame_parse (const char **input, struct btp_location *location)
- int btp_gdb_frame_parse_frame_start (const char **input, uint32_t *number)
- int btp_gdb_frame_parseadd_operator (const char **input, struct btp_strbuf *target)
- int btp_gdb_frame_parse_function_name_chunk (const char **input, bool space_allowed, char **target)
- int btp_gdb_frame_parse_function_name_braces (const char **input, char **target)
- int btp_gdb_frame_parse_function_name_template (const char **input, char **target)
- bool btp_gdb_frame_parse_function_name (const char **input, char **function_name, char **function_type, struct btp_location *location)
- bool btp_gdb_frame_skip_function_args (const char **input, struct btp_location *location)
- bool btp_gdb_frame_parse_function_call (const char **input, char **function_name, char **function_type, struct btp_location *location)
- bool btp_gdb_frame_parse_address_in_function (const char **input, uint64_t *address, char **function_name, char **function_type, struct btp_location *location)
- bool btp_gdb_frame_parse_file_location (const char **input, char **file, uint32_t *file_line, struct btp_location *location)

- struct btp_gdb_frame * btp_gdb_frame_parse_header (const char **input, struct btp_location *location)
- void btp_gdb_frame_remove_func_prefix (struct btp_gdb_frame *frame, const char *prefix, int num)

9.10.1 Detailed Description

Single frame of GDB stack trace thread.

9.10.2 Function Documentation

9.10.2.1 struct btp_gdb_frame* btp_gdb_frame_append (struct btp_gdb_frame * *dest*, struct btp_gdb_frame * *item*) [read]

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

Returns

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

9.10.2.2 void btp_gdb_frame_append_to_str (struct btp_gdb_frame * *frame*, struct btp_strbuf * *dest*, bool *verbose*)

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

Parameters

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

9.10.2.3 bool btp_gdb_frame_calls_func (struct btp_gdb_frame * *frame*, const char * *function_name*, ...)

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

Parameters

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

Returns

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

9.10.2.4 int btp_gdb_frame_cmp (struct btp_gdb_frame * *frame1*, struct btp_gdb_frame * *frame2*, bool *compare_number*)

Compares two frames.

Parameters

<i>frame1</i>	It must be non-NULL pointer. It's not modified by calling this function.
<i>frame2</i>	It must be non-NULL pointer. It's not modified by calling this function.
<i>compare_number</i>	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.

9.10.2.5 `int btp_gdb_frame_cmp_simple (struct btp_gdb_frame * frame1, struct btp_gdb_frame * frame2)`

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

Parameters

<i>frame1</i>	It must be non-NULL pointer. It's not modified by calling this function.
<i>frame2</i>	It must be non-NULL pointer. It's not modified by calling this function.

Returns

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

9.10.2.6 `struct btp_gdb_frame* btp_gdb_frame_dup (struct btp_gdb_frame * frame, bool siblings)` [read]

Creates a duplicate of the frame.

Parameters

<i>frame</i>	It must be non-NULL pointer. The frame is not modified by calling this function.
<i>siblings</i>	Whether to duplicate also siblings referenced by frame->next. If false, frame->next is not duplicated for the new frame, but it is set to NULL.

Returns

This function never returns NULL. The returned duplicate frame must be released by calling the function `btp_gdb_frame_free()`.

9.10.2.7 `void btp_gdb_frame_free (struct btp_gdb_frame * frame)`

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

Parameters

<i>frame</i>	If the frame is NULL, no operation is performed.
--------------	--

9.10.2.8 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.

9.10.2.9 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()`.

9.10.2.10 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

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

9.10.2.11 bool `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

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

9.10.2.12 `bool btb_gdb_frame_parse_file_location (const char ** input, char ** file, uint32_t * file_line, struct btb_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

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

9.10.2.13 `int btb_gdb_frame_parse_frame_start (const char ** input, uint32_t * number)`

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

Returns

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

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

9.10.2.14 `bool btb_gdb_frame_parse_function_call (const char ** input, char ** function_name, char ** function_type, struct btb_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

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

9.10.2.15 `bool btp_gdb_frame_parse_function_name (const char ** input, char ** function_name, char ** function_type, struct btp_location * location)`

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

Parameters

<i>function_name</i>	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 <code>free()</code> . If this function returns true, this pointer is guaranteed to be non-NULL.
<i>location</i>	The caller must provide a pointer to an instance of <code>btp_location</code> here. The line and column members of the location are gradually increased as the parser handles the input, so the location should be initialized before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

Returns

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

9.10.2.16 `int btp_gdb_frame_parse_function_name_braces (const char ** input, char ** target)`

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

Returns

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

9.10.2.17 `int btp_gdb_frame_parse_function_name_chunk (const char ** input, bool space_allowed, char ** target)`

Parses a part of function name from the input.

Parameters

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

Returns

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

9.10.2.18 `int btp_gdb_frame_parse_function_name_template (const char ** input, char ** target)`

Returns

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

9.10.2.19 `struct btp_gdb_frame* btp_gdb_frame_parse_header (const char ** input, struct btp_location * location)` [read]

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

Parameters

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

Returns

Newly created frame struct or NULL. The returned frame struct should be released by btp_gdb_frame_free().

9.10.2.20 `int btp_gdb_frame_parseadd_operator (const char ** input, struct btp_strbuf * target)`

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

Parameters

<i>target</i>	The parsed operator name is appened to the string buffer provided, if an operator is found. Otherwise the string buffer is not changed.
---------------	---

Returns

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

9.10.2.21 `void btp_gdb_frame_remove_func_prefix (struct btp_gdb_frame * frame, const char * prefix, int num)`

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

9.10.2.22 `bool btp_gdb_frame_skip_function_args (const char ** input, struct btp_location * location)`

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

Parameters

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

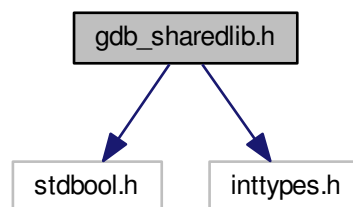
9.11 gdb_sharedlib.h File Reference

Shared library information as produced by GDB.

```
#include <stdbool.h>
```

```
#include <inttypes.h>
```

Include dependency graph for `gdb_sharedlib.h`:



Data Structures

- `struct btp_gdb_sharedlib`
A shared library memory location as reported by GDB.

Enumerations

- `enum { SYMS_OK, SYMS_WRONG, SYMS_NOT_FOUND }`

Functions

- `struct btp_gdb_sharedlib * btp_gdb_sharedlib_new ()`
- `void btp_gdb_sharedlib_init (struct btp_gdb_sharedlib *sharedlib)`
- `void btp_gdb_sharedlib_free (struct btp_gdb_sharedlib *sharedlib)`
- `struct btp_gdb_sharedlib * btp_gdb_sharedlib_append (struct btp_gdb_sharedlib *dest, struct btp_gdb_sharedlib *item)`
- `struct btp_gdb_sharedlib * btp_gdb_sharedlib_dup (struct btp_gdb_sharedlib *sharedlib, bool 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)

9.11.1 Detailed Description

Shared library information as produced by GDB.

9.11.2 Function Documentation

9.11.2.1 struct btp_gdb_sharedlib* btp_gdb_sharedlib_append (struct btp_gdb_sharedlib * *dest*, struct btp_gdb_sharedlib * *item*) [read]

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

Returns

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

9.11.2.2 int btp_gdb_sharedlib_count (struct btp_gdb_sharedlib * *sharedlib*)

Returns the number of sharedlibs in the list.

9.11.2.3 struct btp_gdb_sharedlib* btp_gdb_sharedlib_dup (struct btp_gdb_sharedlib * *sharedlib*, bool *siblings*) [read]

Creates a duplicate of the sharedlib structure.

Parameters

<i>sharedlib</i>	Structure to be duplicated.
<i>siblings</i>	Whether to duplicate a single structure or whole list.

Returns

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

9.11.2.4 struct btp_gdb_sharedlib* btp_gdb_sharedlib_find_address (struct btp_gdb_sharedlib * *first*, uint64_t *address*) [read]

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

Returns

Pointer to an existing structure or NULL if not found.

9.11.2.5 void btp_gdb_sharedlib_free (struct btp_gdb_sharedlib * *sharedlib*)

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

Parameters

<i>sharedlib</i>	If sharedlib is NULL, no operation is performed.
------------------	--

9.11.2.6 void btp_gdb_sharedlib_init (struct btp_gdb_sharedlib * *sharedlib*)

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

9.11.2.7 struct btp_gdb_sharedlib* btp_gdb_sharedlib_new () [read]

Creates and initializes a new sharedlib structure.

Returns

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

9.11.2.8 struct btp_gdb_sharedlib* btp_gdb_sharedlib_parse (const char * *input*) [read]

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

Parameters

<i>input</i>	String representing the stacktrace.
--------------	-------------------------------------

Returns

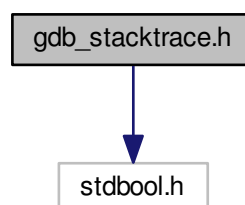
First element of the list of loaded libraries.

9.12 gdb_stacktrace.h File Reference

Stack trace as produced by GDB.

```
#include <stdbool.h>
```

Include dependency graph for gdb_stacktrace.h:



Data Structures

- struct `btg_gdb_stacktrace`
A stack trace produced by GDB.

Functions

- struct `btg_gdb_stacktrace` * `btg_gdb_stacktrace_new` ()
- void `btg_gdb_stacktrace_init` (struct `btg_gdb_stacktrace` *`stacktrace`)
- void `btg_gdb_stacktrace_free` (struct `btg_gdb_stacktrace` *`stacktrace`)
- struct `btg_gdb_stacktrace` * `btg_gdb_stacktrace_dup` (struct `btg_gdb_stacktrace` *`stacktrace`)
- int `btg_gdb_stacktrace_get_thread_count` (struct `btg_gdb_stacktrace` *`stacktrace`)
- void `btg_gdb_stacktrace_remove_threads_except_one` (struct `btg_gdb_stacktrace` *`stacktrace`, struct `btg_gdb_thread` *`thread`)
- struct `btg_gdb_thread` * `btg_gdb_stacktrace_find_crash_thread` (struct `btg_gdb_stacktrace` *`stacktrace`)
- void `btg_gdb_stacktrace_limit_frame_depth` (struct `btg_gdb_stacktrace` *`stacktrace`, int `depth`)
- float `btg_gdb_stacktrace_quality_simple` (struct `btg_gdb_stacktrace` *`stacktrace`)
- float `btg_gdb_stacktrace_quality_complex` (struct `btg_gdb_stacktrace` *`stacktrace`)
- char * `btg_gdb_stacktrace_to_text` (struct `btg_gdb_stacktrace` *`stacktrace`, bool `verbose`)
- struct `btg_gdb_frame` * `btg_gdb_stacktrace_get_crash_frame` (struct `btg_gdb_stacktrace` *`stacktrace`)
- char * `btg_gdb_stacktrace_get_duplication_hash` (struct `btg_gdb_stacktrace` *`stacktrace`)
- struct `btg_gdb_stacktrace` * `btg_gdb_stacktrace_parse` (const char **`input`, struct `btg_location` *`location`)
- bool `btg_gdb_stacktrace_parse_header` (const char **`input`, struct `btg_gdb_frame` **`frame`, struct `btg_location` *`location`)
- void `btg_gdb_stacktrace_set_libnames` (struct `btg_gdb_stacktrace` *`stacktrace`)
- struct `btg_gdb_thread` * `btg_gdb_stacktrace_get_optimized_thread` (struct `btg_gdb_stacktrace` *`stacktrace`, int `max_frames`)

9.12.1 Detailed Description

Stack trace as produced by GDB.

9.12.2 Function Documentation

9.12.2.1 struct `btg_gdb_stacktrace`* `btg_gdb_stacktrace_dup` (struct `btg_gdb_stacktrace` * *stacktrace*) [read]

Creates a duplicate of a stacktrace.

Parameters

<i>stacktrace</i>	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 `btg_gdb_stacktrace_free`().

9.12.2.2 `struct btp_gdb_thread* btp_gdb_stacktrace_find_crash_thread (struct btp_gdb_stacktrace * stacktrace)` [read]

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

Parameters

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

9.12.2.3 `void btp_gdb_stacktrace_free (struct btp_gdb_stacktrace * stacktrace)`

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

Parameters

<i>stacktrace</i>	If the stacktrace is NULL, no operation is performed.
-------------------	---

9.12.2.4 `struct btp_gdb_frame* btp_gdb_stacktrace_get_crash_frame (struct btp_gdb_stacktrace * stacktrace)` [read]

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

Parameters

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

Returns

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

9.12.2.5 `char* btp_gdb_stacktrace_get_duplication_hash (struct btp_gdb_stacktrace * stacktrace)`

Calculates the duplication hash string of the stacktrace.

Parameters

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

Returns

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

9.12.2.6 `struct btp_gdb_thread* btp_gdb_stacktrace_get_optimized_thread (struct btp_gdb_stacktrace * stacktrace, int max_frames)` [read]

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

Parameters

<i>stacktrace</i>	It must be non-NULL pointer. It's not modified by calling this function.
<i>max_frames</i>	The maximum number of frames in the returned crash thread. Superfluous frames are removed from the returned thread.

Returns

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

9.12.2.7 `int btp_gdb_stacktrace_get_thread_count (struct btp_gdb_stacktrace * stacktrace)`

Returns a number of threads in the stacktrace.

Parameters

<i>stacktrace</i>	It's not modified by calling this function.
-------------------	---

9.12.2.8 `void btp_gdb_stacktrace_init (struct btp_gdb_stacktrace * stacktrace)`

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

9.12.2.9 `void btp_gdb_stacktrace_limit_frame_depth (struct btp_gdb_stacktrace * stacktrace, int depth)`

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

Parameters

<i>stacktrace</i>	Must be non-NULL pointer.
-------------------	---------------------------

9.12.2.10 `struct btp_gdb_stacktrace* btp_gdb_stacktrace_new ()` [read]

Creates and initializes a new stack trace structure.

Returns

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

9.12.2.11 `struct btp_gdb_stacktrace* btp_gdb_stacktrace_parse (const char ** input, struct btp_location * location)` [read]

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

```
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

<i>input</i>	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.
<i>location</i>	The caller must provide a pointer to an instance of <code>btp_location</code> 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 <code>btp_location_init()</code> before calling this function to get reasonable values. When this function returns false (an error occurred), the structure will contain the error line, column, and message.

Returns

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

9.12.2.12 `bool btp_gdb_stacktrace_parse_header (const char ** input, struct btp_gdb_frame ** frame, struct btp_location * location)`

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

Parameters

<i>input</i>	Pointer that will be moved to point behind the header if the header is successfully detected and parsed.
--------------	--

<i>frame</i>	<p>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.</p> <pre> [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 </pre>
--------------	---

9.12.2.13 float `btp_gdb_stacktrace_quality_complex` (struct `btp_gdb_stacktrace` * *stacktrace*)

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

Parameters

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

Returns

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

9.12.2.14 float `btp_gdb_stacktrace_quality_simple` (struct `btp_gdb_stacktrace` * *stacktrace*)

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

Parameters

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

Returns

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

9.12.2.15 void `btp_gdb_stacktrace_remove_threads_except_one` (struct `btp_gdb_stacktrace` * *stacktrace*, struct `btp_gdb_thread` * *thread*)

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

Parameters

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

9.12.2.16 void `btg_gdb_stacktrace_set_libnames` (struct `btg_gdb_stacktrace` * *stacktrace*)

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

9.12.2.17 char* `btg_gdb_stacktrace_to_text` (struct `btg_gdb_stacktrace` * *stacktrace*, bool *verbose*)

Returns textual representation of the stacktrace.

Parameters

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

Returns

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

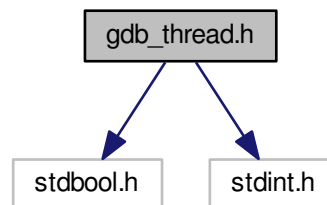
9.13 gdb_thread.h File Reference

Single thread of execution of GDB stack trace.

```
#include <stdbool.h>
```

```
#include <stdint.h>
```

Include dependency graph for `gdb_thread.h`:



Data Structures

- struct `btg_gdb_thread`

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

Functions

- struct `btg_gdb_thread` * `btg_gdb_thread_new` ()
- void `btg_gdb_thread_init` (struct `btg_gdb_thread` **thread*)
- void `btg_gdb_thread_free` (struct `btg_gdb_thread` **thread*)

- struct btp_gdb_thread * btp_gdb_thread_dup (struct btp_gdb_thread *thread, bool siblings)
- int btp_gdb_thread_cmp (struct btp_gdb_thread *thread1, struct btp_gdb_thread *thread2)
- struct btp_gdb_thread * btp_gdb_thread_append (struct btp_gdb_thread *dest, struct btp_gdb_thread *item)
- int btp_gdb_thread_get_frame_count (struct btp_gdb_thread *thread)
- void btp_gdb_thread_quality_counts (struct btp_gdb_thread *thread, int *ok_count, int *all_count)
- float btp_gdb_thread_quality (struct btp_gdb_thread *thread)
- bool btp_gdb_thread_remove_frame (struct btp_gdb_thread *thread, struct btp_gdb_frame *frame)
- bool btp_gdb_thread_remove_frames_above (struct btp_gdb_thread *thread, struct btp_gdb_frame *frame)
- void btp_gdb_thread_remove_frames_below_n (struct btp_gdb_thread *thread, int n)
- void btp_gdb_thread_append_to_str (struct btp_gdb_thread *thread, struct btp_strbuf *dest, bool verbose)
- struct btp_gdb_thread * btp_gdb_thread_parse (const char **input, struct btp_location *location)
- int btp_gdb_thread_skip_lwp (const char **input)
- struct btp_gdb_thread * btp_gdb_thread_parse_funs (const char *input)
- char * btp_gdb_thread_format_funs (struct btp_gdb_thread *thread)

9.13.1 Detailed Description

Single thread of execution of GDB stack trace.

9.13.2 Function Documentation

9.13.2.1 `struct btp_gdb_thread* btp_gdb_thread_append (struct btp_gdb_thread * dest, struct btp_gdb_thread * item)` [read]

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

Returns

This function returns the 'dest' thread.

9.13.2.2 `void btp_gdb_thread_append_to_str (struct btp_gdb_thread * thread, struct btp_strbuf * dest, bool verbose)`

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

9.13.2.3 `int btp_gdb_thread_cmp (struct btp_gdb_thread * thread1, struct btp_gdb_thread * thread2)`

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

Returns

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

9.13.2.4 `struct btp_gdb_thread* btp_gdb_thread_dup (struct btp_gdb_thread * thread, bool siblings)` [read]

Creates a duplicate of the thread.

Parameters

<i>thread</i>	It must be non-NULL pointer. The thread is not modified by calling this function.
<i>siblings</i>	Whether to duplicate also siblings referenced by <code>thread->next</code> . If false, <code>thread->next</code> is not duplicated for the new frame, but it is set to NULL.

9.13.2.5 `char* btp_gdb_thread_format_funs (struct btp_gdb_thread * thread)`

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

Returns

Newly allocated string, which should be released by calling `free()`. The string can be parsed by `btp_gdb_thread_parse_funs()`.

9.13.2.6 `void btp_gdb_thread_free (struct btp_gdb_thread * thread)`

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

Parameters

<i>thread</i>	If thread is NULL, no operation is performed.
---------------	---

9.13.2.7 `int btp_gdb_thread_get_frame_count (struct btp_gdb_thread * thread)`

Returns the number of frames in the thread.

9.13.2.8 `void btp_gdb_thread_init (struct btp_gdb_thread * thread)`

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

9.13.2.9 `struct btp_gdb_thread* btp_gdb_thread_new ()` [read]

Creates and initializes a new frame structure.

Returns

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

9.13.2.10 `struct btp_gdb_thread* btp_gdb_thread_parse (const char ** input, struct btp_location * location)` [read]

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

Parameters

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

Returns

NULL or newly allocated structure, which should be released by calling btp_gdb_thread_free().

9.13.2.11 `struct btp_gdb_thread* btp_gdb_thread_parse_funs (const char * input)` [read]

Create a thread from function and library names.

Parameters

<i>input</i>	String containing function names and library names separated by space, one frame per line.
--------------	--

Returns

Newly allocated structure, which should be released by calling btp_gdb_thread_free().

9.13.2.12 `float btp_gdb_thread_quality (struct btp_gdb_thread * thread)`

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

Parameters

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

Returns

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

9.13.2.13 `void btp_gdb_thread_quality_counts (struct btp_gdb_thread * thread, int * ok_count, int * all_count)`

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

Parameters

<i>ok_count</i>	
<i>all_count</i>	Not zeroed. This function just adds the numbers to ok_count and all_count.

9.13.2.14 `bool btp_gdb_thread_remove_frame (struct btp_gdb_thread * thread, struct btp_gdb_frame * frame)`

Removes the frame from the thread and then deletes it.

Returns

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

9.13.2.15 `bool btp_gdb_thread_remove_frames_above (struct btp_gdb_thread * thread, struct btp_gdb_frame * frame)`

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

Returns

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

9.13.2.16 `void btp_gdb_thread_remove_frames_below_n (struct btp_gdb_thread * thread, int n)`

Keeps only the top n frames in the thread.

9.13.2.17 `int btp_gdb_thread_skip_lwp (const char ** input)`

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

Returns

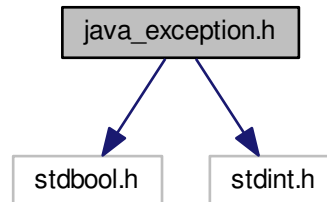
The number of characters parsed from input. 0 if the input does not contain a LWP section.

9.14 java_exception.h File Reference

Single exception of execution of JAVA stack trace.

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


Include dependency graph for java_exception.h:



Data Structures

- struct `btp_java_exception`

A exception of execution of a JAVA-produced stack trace.

Functions

- struct `btp_java_exception` * `btp_java_exception_new` ()
- void `btp_java_exception_init` (struct `btp_java_exception` *`exception`)
- void `btp_java_exception_free` (struct `btp_java_exception` *`exception`)
- bool `btp_java_exception_pop` (struct `btp_java_exception` *`exception`)
- struct `btp_java_exception` * `btp_java_exception_dup` (struct `btp_java_exception` *`exception`, bool `deep`)
- int `btp_java_exception_cmp` (struct `btp_java_exception` *`exception1`, struct `btp_java_exception` *`exception2`, bool `deep`)
- int `btp_java_exception_get_frame_count` (struct `btp_java_exception` *`exception`, bool `deep`)
- void `btp_java_exception_quality_counts` (struct `btp_java_exception` *`exception`, int *`ok_count`, int *`all_count`, bool `deep`)
- float `btp_java_exception_quality` (struct `btp_java_exception` *`exception`, bool `deep`)
- bool `btp_java_exception_remove_frame` (struct `btp_java_exception` *`exception`, struct `btp_java_frame` *`frame`, bool `deep`)
- bool `btp_java_exception_remove_frames_above` (struct `btp_java_exception` *`exception`, struct `btp_java_frame` *`frame`, bool `deep`)
- unsigned `btp_java_exception_remove_frames_below_n` (struct `btp_java_exception` *`exception`, unsigned `n`, bool `deep`)
- void `btp_java_exception_append_to_str` (struct `btp_java_exception` *`exception`, struct `btp_strbuf` *`dest`)
- struct `btp_java_exception` * `btp_java_exception_parse` (const char **`input`, struct `btp_location` *`location`)

9.14.1 Detailed Description

Single exception of execution of JAVA stack trace.

9.14.2 Function Documentation

9.14.2.1 `void btp_java_exception_append_to_str (struct btp_java_exception * exception, struct btp_strbuf * dest)`

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

Parameters

<i>exception</i>	Formatted exception. Non-NULL pointer.
<i>dest</i>	An output buffer. Non-NULL pointer.

9.14.2.2 `int btp_java_exception_cmp (struct btp_java_exception * exception1, struct btp_java_exception * exception2, bool deep)`

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

Returns

Returns 0 if the exceptions 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.

9.14.2.3 `struct btp_java_exception* btp_java_exception_dup (struct btp_java_exception * exception, bool deep)` [read]

Creates a duplicate of the exception.

Parameters

<i>exception</i>	It must be non-NULL pointer. The exception is not modified by calling this function.
<i>deep</i>	Whether to duplicate also the inner exception. If false, exception->inner is not duplicated for the new frame, but it is set to NULL.

9.14.2.4 `void btp_java_exception_free (struct btp_java_exception * exception)`

Releases the memory held by the exception including inner exception.

Parameters

<i>exception</i>	If exception is NULL, no operation is performed.
------------------	--

9.14.2.5 `int btp_java_exception_get_frame_count (struct btp_java_exception * exception, bool deep)`

Returns the number of frames in the exception.

9.14.2.6 void `btjava_exception_init` (struct `btjava_exception` * *exception*)

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

9.14.2.7 struct `btjava_exception`* `btjava_exception_new` () [read]

Creates and initializes a new exception structure.

Returns

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

9.14.2.8 struct `btjava_exception`* `btjava_exception_parse` (const char ** *input*, struct `bt_location` * *location*) [read]

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

Parameters

<i>location</i>	The caller must provide a pointer to struct <code>bt_location</code> 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 `btjava_exception_free()`.

9.14.2.9 bool `btjava_exception_pop` (struct `btjava_exception` * *exception*)

Replaces exception by its inner exception if the exception has one.

Parameters

<i>exception</i>	It must be non-NULL pointer. The exception is not modified by calling this function.
------------------	--

Returns

TRUE if exceptions was popped, otherwise FALSE

9.14.2.10 float `btjava_exception_quality` (struct `btjava_exception` * *exception*, bool *deep*)

Returns the quality of the exception. 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

<i>exception</i>	Must be a non-NULL pointer. It's not modified in this function.
<i>deep</i>	If logical true, work out the quality from inner exceptions too.

Returns

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

9.14.2.11 `void btp_java_exception_quality_counts (struct btp_java_exception * exception, int * ok_count, int * all_count, bool deep)`

Counts the number of 'good' frames and the number of all frames in a exception. Good means that the function name is known (so it's not just 'Unknown Source' | 'Native method').

Parameters

<i>ok_count</i>	
<i>all_count</i>	
<i>deep</i>	If logical true, work out the sum from inner exceptions too. Not zeroed. This function just adds the numbers to <i>ok_count</i> and <i>all_count</i> .

9.14.2.12 `bool btp_java_exception_remove_frame (struct btp_java_exception * exception, struct btp_java_frame * frame, bool deep)`

Removes the frame from the exception and then deletes it.

Parameters

<i>exception</i>	Modified exception. Non-NULL pointer.
<i>frame</i>	Removed frame. Non-NULL pointer.
<i>deep</i>	If logical true, remove the frame from inner exception if frame is not found in the current one.

Returns

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

9.14.2.13 `bool btp_java_exception_remove_frames_above (struct btp_java_exception * exception, struct btp_java_frame * frame, bool deep)`

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

Parameters

<i>exception</i>	Modified exception. Non-NULL pointer.
<i>frame</i>	A new topmost frame. Non-NULL pointer.
<i>deep</i>	If logical true, remove inner exception because an inner exception is always higher in stack trace.

Returns

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

9.14.2.14 `unsigned btp_java_exception_remove_frames_below_n (struct btp_java_exception * exception, unsigned n, bool deep)`

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

Parameters

<i>exception</i>	Modified exception. Non-NULL pointer.
<i>n</i>	A number of left frames
<i>deep</i>	If logical true, take inner exceptions into account. It can leads to replacement of the passed exception byt its inner excpetion.

Returns

if count of frames is shorter then *n* returns the differences; * otherwise 0

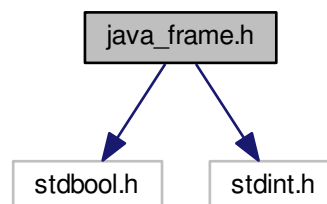
9.15 java_frame.h File Reference

java frame structure and related algorithms.

```
#include <stdbool.h>
```

```
#include <stdint.h>
```

Include dependency graph for java_frame.h:

**Data Structures**

- `struct btp_java_frame`

Functions

- `struct btp_java_frame * btp_java_frame_new ()`

- void `btp_java_frame_init` (struct `btp_java_frame` *`frame`)
- void `btp_java_frame_free` (struct `btp_java_frame` *`frame`)
- struct `btp_java_frame` * `btp_java_frame_dup` (struct `btp_java_frame` *`frame`, bool `siblings`)
- int `btp_java_frame_cmp` (struct `btp_java_frame` *`frame1`, struct `btp_java_frame` *`frame2`)
- void `btp_java_frame_append_to_str` (struct `btp_java_frame` *`frame`, struct `btp_strbuf` *`dest`)
- struct `btp_java_frame` * `btp_java_frame_parse` (const char **`input`, struct `btp_location` *`location`)

9.15.1 Detailed Description

java frame structure and related algorithms.

9.15.2 Function Documentation

9.15.2.1 void `btp_java_frame_append_to_str` (struct `btp_java_frame` * *frame*, struct `btp_strbuf` * *dest*)

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

Parameters

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

9.15.2.2 int `btp_java_frame_cmp` (struct `btp_java_frame` * *frame1*, struct `btp_java_frame` * *frame2*)

Compares two frames.

Parameters

<i>frame1</i>	It must be non-NULL pointer. It's not modified by calling this function.
<i>frame2</i>	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`.

9.15.2.3 struct `btp_java_frame`* `btp_java_frame_dup` (struct `btp_java_frame` * *frame*, bool *siblings*) [read]

Creates a duplicate of the frame.

Parameters

<i>frame</i>	It must be non-NULL pointer. The frame is not modified by calling this function.
<i>siblings</i>	Whether to duplicate also siblings referenced by <code>frame->next</code> . If false, <code>frame->next</code> 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_java_frame_free()`.

9.15.2.4 void btp_java_frame_free (struct btp_java_frame * *frame*)

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

Parameters

<i>frame</i>	If the frame is NULL, no operation is performed.
--------------	--

9.15.2.5 void btp_java_frame_init (struct btp_java_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.

9.15.2.6 struct btp_java_frame* btp_java_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_java_frame_free()`.

9.15.2.7 struct btp_java_frame* btp_java_frame_parse (const char ** *input*, struct btp_location * *location*) [read]

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

Returns

Allocated pointer with a frame structure. The pointer should be released by `btp_java_frame_free()`.

Parameters

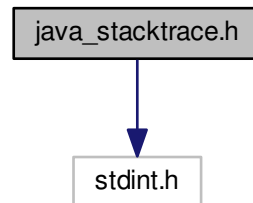
<i>location</i>	The caller must provide a pointer to an instance of <code>btp_location</code> here. When this function returns NULL, the structure will contain the error line, column, and message. The line and column members of the location are gradually increased as the parser handles the input, so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

9.16 java_stacktrace.h File Reference

java stack trace structure and related algorithms.

```
#include <stdint.h>
```

Include dependency graph for java_stacktrace.h:



Data Structures

- struct `btp_java_stacktrace`

Functions

- struct `btp_java_stacktrace` * `btp_java_stacktrace_new` ()
- void `btp_java_stacktrace_init` (struct `btp_java_stacktrace` *`stacktrace`)
- void `btp_java_stacktrace_free` (struct `btp_java_stacktrace` *`stacktrace`)
- struct `btp_java_stacktrace` * `btp_java_stacktrace_dup` (struct `btp_java_stacktrace` *`stacktrace`)
- int `btp_java_stacktrace_cmp` (struct `btp_java_stacktrace` *`stacktrace1`, struct `btp_java_stacktrace` *`stacktrace2`)
- struct `btp_java_stacktrace` * `btp_java_stacktrace_parse` (const char **`input`, struct `btp_location` *`location`)

9.16.1 Detailed Description

java stack trace structure and related algorithms.

9.16.2 Function Documentation

9.16.2.1 int `btp_java_stacktrace_cmp` (struct `btp_java_stacktrace` * *stacktrace1*, struct `btp_java_stacktrace` * *stacktrace2*)

Compares two stacktraces.

Returns

Returns 0 if the stacktraces 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.

9.16.2.2 `struct btp_java_stacktrace* btp_java_stacktrace_dup (struct btp_java_stacktrace * stacktrace)` [read]

Creates a duplicate of the stacktrace.

Parameters

<i>stacktrace</i>	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_java_stacktrace_free()`.

9.16.2.3 `void btp_java_stacktrace_free (struct btp_java_stacktrace * stacktrace)`

Releases the memory held by the stacktrace and its frames.

Parameters

<i>stacktrace</i>	If the stacktrace is NULL, no operation is performed.
-------------------	---

9.16.2.4 `void btp_java_stacktrace_init (struct btp_java_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.

9.16.2.5 `struct btp_java_stacktrace* btp_java_stacktrace_new ()` [read]

Creates and initializes a new stacktrace structure.

Returns

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

9.16.2.6 `struct btp_java_stacktrace* btp_java_stacktrace_parse (const char ** input, struct btp_location * location)` [read]

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

Parameters

<i>input</i>	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.
<i>location</i>	The caller must provide a pointer to an instance of <code>btp_location</code> 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 <code>btp_location_init()</code> 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_java_stacktrace_free()`.

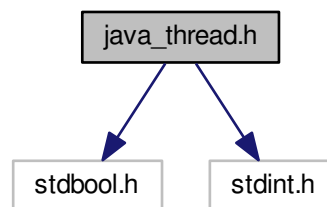
9.17 java_thread.h File Reference

Single thread of execution of JAVA stack trace.

```
#include <stdbool.h>
```

```
#include <stdint.h>
```

Include dependency graph for `java_thread.h`:



Data Structures

- `struct btp_java_thread`

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

Functions

- `struct btp_java_thread * btp_java_thread_new ()`
- `void btp_java_thread_init (struct btp_java_thread *thread)`
- `void btp_java_thread_free (struct btp_java_thread *thread)`
- `struct btp_java_thread * btp_java_thread_dup (struct btp_java_thread *thread, bool siblings)`
- `int btp_java_thread_cmp (struct btp_java_thread *thread1, struct btp_java_thread *thread2)`
- `struct btp_java_thread * btp_java_thread_append (struct btp_java_thread *dest, struct btp_java_thread *item)`
- `int btp_java_thread_get_frame_count (struct btp_java_thread *thread)`
- `void btp_java_thread_quality_counts (struct btp_java_thread *thread, int *ok_count, int *all_count)`
- `float btp_java_thread_quality (struct btp_java_thread *thread)`
- `bool btp_java_thread_remove_frame (struct btp_java_thread *thread, struct btp_java_frame *frame)`
- `bool btp_java_thread_remove_frames_above (struct btp_java_thread *thread, struct btp_java_frame *frame)`
- `void btp_java_thread_remove_frames_below_n (struct btp_java_thread *thread, int n)`
- `void btp_java_thread_append_to_str (struct btp_java_thread *thread, struct btp_strbuf *dest)`

- struct `btpt_java_thread` * `btpt_java_thread_parse` (const char **input, struct `btpt_location` *location)
- struct `btpt_java_thread` * `btpt_java_thread_parse_funs` (const char *input)
- char * `btpt_java_thread_format_funs` (struct `btpt_java_thread` *thread)

9.17.1 Detailed Description

Single thread of execution of JAVA stack trace.

9.17.2 Function Documentation

9.17.2.1 struct `btpt_java_thread`* `btpt_java_thread_append` (struct `btpt_java_thread` * *dest*, struct `btpt_java_thread` * *item*) [read]

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

Returns

This function returns the 'dest' thread.

9.17.2.2 void `btpt_java_thread_append_to_str` (struct `btpt_java_thread` * *thread*, struct `btpt_strbuf` * *dest*)

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

9.17.2.3 int `btpt_java_thread_cmp` (struct `btpt_java_thread` * *thread1*, struct `btpt_java_thread` * *thread2*)

Compares two threads. When comparing the threads, it compares also their exceptions.

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.

9.17.2.4 struct `btpt_java_thread`* `btpt_java_thread_dup` (struct `btpt_java_thread` * *thread*, bool *siblings*) [read]

Creates a duplicate of the thread.

Parameters

<i>thread</i>	It must be non-NULL pointer. The thread is not modified by calling this function.
<i>siblings</i>	Whether to duplicate also siblings referenced by thread->next. If false, thread->next is not duplicated for the new exception, but it is set to NULL.

9.17.2.5 char* `btpt_java_thread_format_funs` (struct `btpt_java_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 `btplib_java_thread_parse_funs()`.

9.17.2.6 void btplib_java_thread_free (struct btplib_java_thread * *thread*)

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

Parameters

<i>thread</i>	If <i>thread</i> is NULL, no operation is performed.
---------------	--

9.17.2.7 int btplib_java_thread_get_frame_count (struct btplib_java_thread * *thread*)

Returns the number of frames in the thread.

9.17.2.8 void btplib_java_thread_init (struct btplib_java_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.

9.17.2.9 struct btplib_java_thread* btplib_java_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 `btplib_java_thread_free()`.

9.17.2.10 struct btplib_java_thread* btplib_java_thread_parse (const char ** *input*, struct btplib_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

<i>location</i>	The caller must provide a pointer to struct <code>btplib_location</code> 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 `btplib_java_thread_free()`.

9.17.2.11 struct btp_java_thread* btp_java_thread_parse_funs (const char * *input*) [read]

Create a thread from function and library names.

Parameters

<i>input</i>	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_java_thread_free().

9.17.2.12 float btp_java_thread_quality (struct btp_java_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

<i>thread</i>	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.

9.17.2.13 void btp_java_thread_quality_counts (struct btp_java_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

<i>ok_count</i>	
<i>all_count</i>	Not zeroed. This function just adds the numbers to ok_count and all_count.

9.17.2.14 bool btp_java_thread_remove_frame (struct btp_java_thread * *thread*, struct btp_java_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.

9.17.2.15 `bool btp_java_thread_remove_frames_above (struct btp_java_thread * thread, struct btp_java_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.

9.17.2.16 `void btp_java_thread_remove_frames_below_n (struct btp_java_thread * thread, int n)`

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

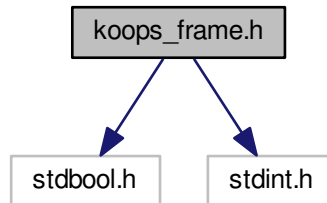
9.18 koops_frame.h File Reference

Kernel oops stack frame.

```
#include <stdbool.h>
```

```
#include <stdint.h>
```

Include dependency graph for koops_frame.h:



Data Structures

- `struct btp_koops_frame`
Kernel oops stack frame.

Functions

- `struct btp_koops_frame * btp_koops_frame_new ()`
- `void btp_koops_frame_init (struct btp_koops_frame *frame)`
- `void btp_koops_frame_free (struct btp_koops_frame *frame)`
- `struct btp_koops_frame * btp_koops_frame_dup (struct btp_koops_frame *frame, bool siblings)`
- `int btp_koops_frame_cmp (struct btp_koops_frame *frame1, struct btp_koops_frame *frame2)`

- `struct btp_koops_frame * btp_koops_frame_append (struct btp_koops_frame *dest, struct btp_koops_frame *item)`
- `struct btp_koops_frame * btp_koops_frame_parse (const char **input)`
- `bool btp_koops_skip_timestamp (const char **input)`
- `bool btp_koops_parse_address (const char **input, uint64_t *address)`
- `bool btp_koops_parse_module_name (const char **input, char **module_name)`
- `bool btp_koops_parse_function (const char **input, char **function_name, uint64_t *function_offset, uint64_t *function_length, char **module_name)`
- `char * btp_koops_frame_to_json (struct btp_koops_frame *frame)`

9.18.1 Detailed Description

Kernel oops stack frame.

9.18.2 Function Documentation

9.18.2.1 `struct btp_koops_frame* btp_koops_frame_append (struct btp_koops_frame * dest, struct btp_koops_frame * item)` [read]

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

Returns

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

9.18.2.2 `int btp_koops_frame_cmp (struct btp_koops_frame * frame1, struct btp_koops_frame * frame2)`

Compares two frames.

Parameters

<i>frame1</i>	It must be non-NULL pointer. It's not modified by calling this function.
<i>frame2</i>	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.

9.18.2.3 `struct btp_koops_frame* btp_koops_frame_dup (struct btp_koops_frame * frame, bool siblings)` [read]

Creates a duplicate of the frame.

Parameters

<i>frame</i>	It must be non-NULL pointer. The frame is not modified by calling this function.
<i>siblings</i>	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()`.

9.18.2.4 void btp_koops_frame_free (struct btp_koops_frame * *frame*)

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

Parameters

<i>frame</i>	If the frame is NULL, no operation is performed.
--------------	--

9.18.2.5 void btp_koops_frame_init (struct btp_koops_frame * *frame*)

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

9.18.2.6 struct btp_koops_frame* btp_koops_frame_new () [read]

Creates and initializes a new frame structure.

Returns

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

9.18.2.7 char* btp_koops_frame_to_json (struct btp_koops_frame * *frame*)

Returns a textual representation of the frame.

Parameters

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

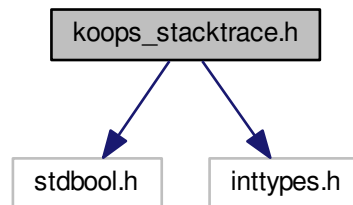
9.19 koops_stacktrace.h File Reference

Kernel oops stack trace structure and related algorithms.

```
#include <stdbool.h>
```

```
#include <inttypes.h>
```


Include dependency graph for `koops_stacktrace.h`:



Data Structures

- `struct btp_koops_stacktrace`

Functions

- `struct btp_koops_stacktrace * btp_koops_stacktrace_new ()`
- `void btp_koops_stacktrace_init (struct btp_koops_stacktrace *stacktrace)`
- `void btp_koops_stacktrace_free (struct btp_koops_stacktrace *stacktrace)`
- `struct btp_koops_stacktrace * btp_koops_stacktrace_dup (struct btp_koops_stacktrace *stacktrace)`
- `int btp_koops_stacktrace_get_frame_count (struct btp_koops_stacktrace *stacktrace)`
- `bool btp_koops_stacktrace_remove_frame (struct btp_koops_stacktrace *stacktrace, struct btp_koops-_frame *frame)`
- `struct btp_koops_stacktrace * btp_koops_stacktrace_parse (const char **input, struct btp_location *location)`
- `char ** btp_koops_stacktrace_parse_modules (const char **input)`

9.19.1 Detailed Description

Kernel oops stack trace structure and related algorithms.

9.19.2 Function Documentation

9.19.2.1 `struct btp_koops_stacktrace* btp_koops_stacktrace_dup (struct btp_koops_stacktrace * stacktrace)` [read]

Creates a duplicate of a stacktrace.

Parameters

<i>stacktrace</i>	The stacktrace to be copied. It's not modified by this function.
-------------------	--

Returns

This function never returns NULL. The returned duplicate must be released by calling the function `btp_koops_stacktrace_free()`.

9.19.2.2 void btp_koops_stacktrace_free (struct btp_koops_stacktrace * *stacktrace*)

Releases the memory held by the stacktrace.

Parameters

<i>stacktrace</i>	If the stacktrace is NULL, no operation is performed.
-------------------	---

9.19.2.3 int btp_koops_stacktrace_get_frame_count (struct btp_koops_stacktrace * *stacktrace*)

Returns the number of frames in the Kerneloops stacktrace.

9.19.2.4 void btp_koops_stacktrace_init (struct btp_koops_stacktrace * *stacktrace*)

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

9.19.2.5 struct btp_koops_stacktrace* btp_koops_stacktrace_new () [read]

Creates and initializes a new stack trace structure.

Returns

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

9.19.2.6 struct btp_koops_stacktrace* btp_koops_stacktrace_parse (const char ** *input*, struct btp_location * *location*) [read]

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

Parameters

<i>input</i>	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.
--------------	--

9.19.2.7 bool btp_koops_stacktrace_remove_frame (struct btp_koops_stacktrace * *stacktrace*, struct btp_koops_frame * *frame*)

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

Returns

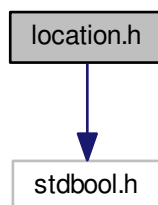
True if the frame was found in the thread and removed and deleted. False if the frame was not found in the thread.

9.20 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)

9.20.1 Detailed Description

Parser location in input file.

9.20.2 Function Documentation

9.20.2.1 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

<i>location</i>	The structure to be modified. It must be a valid pointer.
<i>add_line</i>	Starts from 1. It means that if <code>add_line</code> is 1, the line member of the location structure is not changed.
<i>add_column</i>	Starts from 0.

9.20.2.2 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

<i>add_line</i>	Starts from 1. It means that if <code>add_line</code> is 1, the line is not changed.
<i>add_column</i>	Starts from 0.

9.20.2.3 int `btp_location_cmp` (struct `btp_location` * *location1*, struct `btp_location` * *location2*, bool *compare_messages*)

Compare two locations.

Parameters

<i>location1</i>	It must be non-NULL pointer. It's not modified by calling this function.
<i>location2</i>	It must be non-NULL pointer. It's not modified by calling this function.
<i>compare_messages</i>	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.

9.20.2.4 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.

9.20.2.5 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

<i>line</i>	Must be a valid pointer.
<i>column</i>	Must be a valid pointer.

9.20.2.6 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.

9.20.2.7 char* `btp_location_to_string` (struct `btp_location` * *location*)

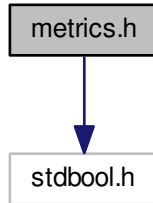
Creates a string representation of location. User must delete the returned string using `free()`.

9.21 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)

9.21.1 Detailed Description

Distance between stack trace threads.

9.21.2 Typedef Documentation

9.21.2.1 typedef float(* `btp_dist_thread_type`)(struct `btp_gdb_thread` *, struct `btp_gdb_thread` *)

A function which compares two threads.

9.21.3 Function Documentation

9.21.3.1 struct `btp_distances`* `btp_distances_dup` (struct `btp_distances` * *distances*) [read]

Creates a duplicate of the distances structure.

Parameters

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

Returns

This function never returns NULL.

9.21.3.2 void `btp_distances_free` (struct `btp_distances` * *distances*)

Releases the memory held by the distances structure.

Parameters

<i>distances</i>	If the distances is NULL, no operation is performed.
------------------	--

9.21.3.3 float `btp_distances_get_distance` (struct `btp_distances` * *distances*, int *i*, int *j*)

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

Parameters

<i>distances</i>	It must be non-NULL pointer.
<i>i</i>	Row in the matrix.
<i>j</i>	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.

9.21.3.4 struct `btp_distances`* `btp_distances_new` (int *m*, int *n*) [read]

Creates and initializes a new distances structure.

Parameters

<i>m</i>	Number of rows.
<i>n</i>	Number of columns.

Returns

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

9.21.3.5 void `btp_distances_set_distance` (struct `btp_distances` * *distances*, int *i*, int *j*, float *d*)

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

Parameters

<i>distances</i>	It must be non-NULL pointer.
<i>i</i>	Row in the matrix.
<i>j</i>	Column in the matrix.
<i>d</i>	Distance.

9.21.3.6 struct `btp_distances`* `btp_gdb_threads_compare` (struct `btp_gdb_thread` ** *threads*, int *m*, int *n*, `btp_dist_thread_type` *dist_func*) [read]

Creates a distances structure by comparing threads.

Parameters

<i>threads</i>	Array of threads. They are not modified by calling this function.
<i>m</i>	Compare first m threads from the array with other threads.
<i>n</i>	Number of threads in the passed array.
<i>dist_func</i>	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.

9.22 normalize.h File Reference

Normalization of stack traces.

Functions

- void `btp_normalize_gdb_thread` (struct `btp_gdb_thread` *thread)
- void `btp_normalize_gdb_stacktrace` (struct `btp_gdb_stacktrace` *stacktrace)
- void `btp_normalize_koops_stacktrace` (struct `btp_koops_stacktrace` *stacktrace)
- struct `btp_gdb_frame` * `btp_glibc_thread_find_exit_frame` (struct `btp_gdb_thread` *thread)

- void `btp_normalize_gdb_paired_unknown_function_names` (struct `btp_gdb_thread` *`thread1`, struct `btp_gdb_thread` *`thread2`)
- void `btp_gdb_normalize_optimize_thread` (struct `btp_gdb_thread` *`thread`)

9.22.1 Detailed Description

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

9.22.2 Function Documentation

9.22.2.1 void `btp_gdb_normalize_optimize_thread` (struct `btp_gdb_thread` * *thread*)

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

9.22.2.2 struct `btp_gdb_frame`* `btp_glibc_thread_find_exit_frame` (struct `btp_gdb_thread` * *thread*) [read]

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

Returns

Returns NULL if such a frame is not found.

9.22.2.3 void `btp_normalize_gdb_paired_unknown_function_names` (struct `btp_gdb_thread` * *thread1*, struct `btp_gdb_thread` * *thread2*)

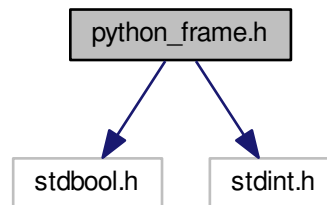
Renames unknown function names ("??") that are between the same function names to be treated as similar in later comparison. Leaves unpair unknown functions unchanged.

9.23 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 `bt_python_frame`

Functions

- struct `bt_python_frame` * `bt_python_frame_new` ()
- void `bt_python_frame_init` (struct `bt_python_frame` *`frame`)
- void `bt_python_frame_free` (struct `bt_python_frame` *`frame`)
- struct `bt_python_frame` * `bt_python_frame_dup` (struct `bt_python_frame` *`frame`, bool `siblings`)

9.23.1 Detailed Description

Python frame structure and related algorithms.

9.23.2 Function Documentation

9.23.2.1 struct `bt_python_frame`* `bt_python_frame_dup` (struct `bt_python_frame` * *frame*, bool *siblings*) [read]

Creates a duplicate of the frame.

Parameters

<i>frame</i>	It must be non-NULL pointer. The frame is not modified by calling this function.
<i>siblings</i>	Whether to duplicate also siblings referenced by <code>frame->next</code> . If false, <code>frame->next</code> 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 `bt_python_frame_free`().

9.23.2.2 void `btp_python_frame_free` (struct `btp_python_frame` * *frame*)

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

Parameters

<i>frame</i>	If the frame is NULL, no operation is performed.
--------------	--

9.23.2.3 void `btp_python_frame_init` (struct `btp_python_frame` * *frame*)

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

9.23.2.4 struct `btp_python_frame`* `btp_python_frame_new` () [read]

Creates and initializes a new frame structure.

Returns

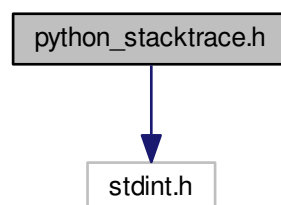
It never returns NULL. The returned pointer must be released by calling the function `btp_python_frame_free()`.

9.24 python_stacktrace.h File Reference

Python stack trace structure and related algorithms.

#include <stdint.h>

Include dependency graph for `python_stacktrace.h`:

**Data Structures**

- struct `btp_python_stacktrace`

Functions

- `struct btp_python_stacktrace * btp_python_stacktrace_new ()`
- `void btp_python_stacktrace_init (struct btp_python_stacktrace *stacktrace)`
- `void btp_python_stacktrace_free (struct btp_python_stacktrace *stacktrace)`
- `struct btp_python_stacktrace * btp_python_stacktrace_dup (struct btp_python_stacktrace *stacktrace)`

9.24.1 Detailed Description

Python stack trace structure and related algorithms.

9.24.2 Function Documentation

9.24.2.1 `struct btp_python_stacktrace* btp_python_stacktrace_dup (struct btp_python_stacktrace * stacktrace)` [read]

Creates a duplicate of the stacktrace.

Parameters

<i>stacktrace</i>	The stacktrace to be copied. It's not modified by this function.
-------------------	--

Returns

This function never returns NULL. The returned duplicate must be released by calling the function `btp_python_stacktrace_free()`.

9.24.2.2 `void btp_python_stacktrace_free (struct btp_python_stacktrace * stacktrace)`

Releases the memory held by the stacktrace and its frames.

Parameters

<i>stacktrace</i>	If the stacktrace is NULL, no operation is performed.
-------------------	---

9.24.2.3 `void btp_python_stacktrace_init (struct btp_python_stacktrace * stacktrace)`

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

9.24.2.4 `struct btp_python_stacktrace* btp_python_stacktrace_new ()` [read]

Creates and initializes a new stacktrace structure.

Returns

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

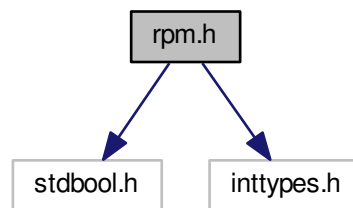
9.25 rpm.h File Reference

RPM-related structures and utilities.

```
#include <stdbool.h>
```

```
#include <inttypes.h>
```

Include dependency graph for rpm.h:



Data Structures

- struct `btp_rpm_consistency`
- struct `btp_rpm_package`

Functions

- struct `btp_rpm_package` * **btp_rpm_package_new** ()
- void **btp_rpm_package_init** (struct `btp_rpm_package` *package)
- void **btp_rpm_package_free** (struct `btp_rpm_package` *package, bool recursive)
- struct `btp_rpm_package` * **btp_rpm_package_append** (struct `btp_rpm_package` *dest, struct `btp_rpm_package` *item)
- struct `btp_rpm_package` * **btp_rpm_package_get_by_name** (const char *name, char **error_message)
- struct `btp_rpm_package` * **btp_rpm_package_get_by_path** (const char *path, char **error_message)
- struct `btp_rpm_consistency` * **btp_rpm_consistency_new** ()
- void **btp_rpm_consistency_init** (struct `btp_rpm_consistency` *consistency)
- void **btp_rpm_consistency_free** (struct `btp_rpm_consistency` *consistency, bool recursive)

9.25.1 Detailed Description

RPM-related structures and utilities.

9.25.2 Function Documentation

9.25.2.1 struct `btp_rpm_package`* **btp_rpm_package_append** (struct `btp_rpm_package` * *dest*, struct `btp_rpm_package` * *item*) [read]

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

Returns

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

9.26 sha1.h File Reference

An implementation of SHA-1 cryptographic hash function.

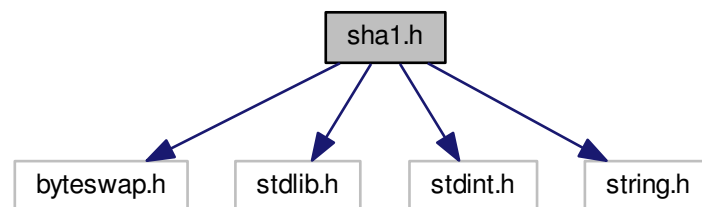
```
#include <byteswap.h>
```

```
#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.

Macros

- #define **BTP_SHA1_RESULT_BIN_LEN** (5 * 4)
- #define **BTP_SHA1_RESULT_LEN** (5 * 4 * 2 + 1)

Functions

- void **btp_sha1_begin** (struct btp_sha1_state *state)
- void **btp_sha1_hash** (struct btp_sha1_state *state, const void *buffer, size_t len)
- void **btp_sha1_end** (struct btp_sha1_state *state, void *resbuf)

9.26.1 Detailed Description

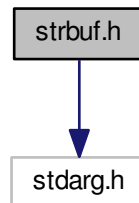
An implementation of SHA-1 cryptographic hash function.

9.27 strbuf.h File Reference

A string buffer structure and related algorithms.

```
#include <stdarg.h>
```

Include dependency graph for strbuf.h:



Data Structures

- struct btp_strbuf
A resizable string buffer.

Functions

- struct btp_strbuf * btp_strbuf_new ()
- void btp_strbuf_init (struct btp_strbuf *strbuf)
- void btp_strbuf_free (struct btp_strbuf *strbuf)
- char * btp_strbuf_free_nobuf (struct btp_strbuf *strbuf)
- void btp_strbuf_clear (struct btp_strbuf *strbuf)
- void btp_strbuf_grow (struct btp_strbuf *strbuf, int num)
- struct btp_strbuf * btp_strbuf_append_char (struct btp_strbuf *strbuf, char c)
- struct btp_strbuf * btp_strbuf_append_str (struct btp_strbuf *strbuf, const char *str)
- struct btp_strbuf * btp_strbuf_prepend_str (struct btp_strbuf *strbuf, const char *str)
- struct btp_strbuf * btp_strbuf_append_strf (struct btp_strbuf *strbuf, const char *format,...)
- struct btp_strbuf * btp_strbuf_append_strfv (struct btp_strbuf *strbuf, const char *format, va_list p)
- struct btp_strbuf * btp_strbuf_prepend_strf (struct btp_strbuf *strbuf, const char *format,...)
- struct btp_strbuf * btp_strbuf_prepend_strfv (struct btp_strbuf *strbuf, const char *format, va_list p)

9.27.1 Detailed Description

A string buffer structure and related algorithms.

9.27.2 Function Documentation

9.27.2.1 `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.

9.27.2.2 `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.

9.27.2.3 `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.

9.27.2.4 `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.

9.27.2.5 `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.

9.27.2.6 `void btp_strbuf_free (struct btp_strbuf * strbuf)`

Releases the memory held by the string buffer.

Parameters

<i>strbuf</i>	If the <i>strbuf</i> is NULL, no operation is performed.
---------------	--

9.27.2.7 `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()`.

9.27.2.8 `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`.

9.27.2.9 `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.

9.27.2.10 `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()`.

9.27.2.11 `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.

9.27.2.12 `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.

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

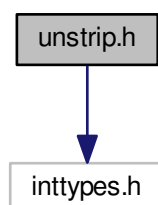
Same as `btp_strbuf_prepend_strf` except that `va_list` is used instead of variable number of arguments.

9.28 unstrip.h File Reference

Parser for the output of the unstrip utility.

`#include <inttypes.h>`

Include dependency graph for unstrip.h:



Data Structures

- `struct btp_unstrip_entry`

Core dump memory layout as reported by the unstrip utility.

Functions

- struct btp_unstrip_entry * **btp_unstrip_parse** (const char *unstrip_output)
- struct btp_unstrip_entry * **btp_unstrip_find_address** (struct btp_unstrip_entry *entries, uint64_t address)
- void **btp_unstrip_free** (struct btp_unstrip_entry *entries)

9.28.1 Detailed Description

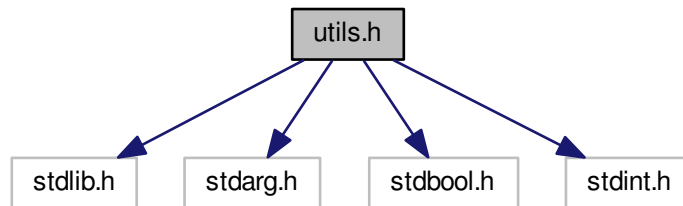
Parser for the output of the unstrip utility.

9.29 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:



Macros

- #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_strchr_location` (const char *s, int c, int *line, int *column)
- char * `btp_strstr_location` (const char *haystack, const char *needle, int *line, int *column)
- size_t `btp_strspn_location` (const char *s, const char *accept, int *line, int *column)
- char * `btp_file_to_string` (const char *filename)
- bool `btp_skip_char` (const char **input, char c)
- bool `btp_skip_char_limited` (const char **input, const char *allowed)
- bool `btp_parse_char_limited` (const char **input, const char *allowed, char *result)
- int `btp_skip_char_sequence` (const char **input, char c)
- int `btp_skip_char_span` (const char **input, const char *chars)
- int `btp_skip_char_span_location` (const char **input, const char *chars, int *line, int *column)
- int `btp_parse_char_span` (const char **input, const char *accept, char **result)
- int `btp_skip_char_cspan` (const char **input, const char *reject)
- bool `btp_parse_char_cspan` (const char **input, const char *reject, char **result)
- int `btp_skip_string` (const char **input, const char *string)
- bool `btp_parse_string` (const char **input, const char *string, char **result)
- char `btp_parse_digit` (const char **input)
- int `btp_skip_uint` (const char **input)
- int `btp_parse_uint32` (const char **input, uint32_t *result)
- int **`btp_parse_uint64`** (const char **input, uint64_t *result)
- int `btp_skip_hexadecimal_uint` (const char **input)
- int `btp_skip_hexadecimal_0xuint` (const char **input)
- int `btp_parse_hexadecimal_uint64` (const char **input, uint64_t *result)
- int `btp_parse_hexadecimal_0xuint64` (const char **input, uint64_t *result)
- char * **`btp_skip_whitespace`** (const char *s)
- char * **`btp_skip_non_whitespace`** (const char *s)
- char * `btp_bin2hex` (char *dst, const char *str, int count)
- char * **`btp_indent`** (const char *input, int spaces)
- char * **`btp_indent_except_first_line`** (const char *input, int spaces)

Variables

- bool `btp_debug_parser`

9.29.1 Detailed Description

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

9.29.2 Function Documentation

9.29.2.1 char* `btp_asprintf` (const char * *format*, ...)

Never returns NULL.

9.29.2.2 char* btp_bin2hex (char * *dst*, const char * *str*, int *count*)

Emit a string of hex representation of bytes.

9.29.2.3 char* btp_file_to_string (const char * *filename*)

Loads file contents to a string.

Returns

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

9.29.2.4 void* btp_malloc (size_t *size*)

Never returns NULL.

9.29.2.5 void* btp_mallocz (size_t *size*)

Never returns NULL.

9.29.2.6 bool btp_parse_char_cspan (const char ** *input*, const char * *reject*, char ** *result*)

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

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

9.29.2.7 bool btp_parse_char_limited (const char ** *input*, const char * *allowed*, char * *result*)

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

9.29.2.8 int btp_parse_char_span (const char ** *input*, const char * *accept*, char ** *result*)

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

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

9.29.2.9 char btp_parse_digit (const char ** *input*)

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

9.29.2.10 int btp_parse_hexadecimal_0xuint64 (const char ** *input*, uint64_t * *result*)

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

Returns

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

9.29.2.11 int btp_parse_hexadecimal_uint64 (const char ** *input*, uint64_t * *result*)

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

Returns

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

9.29.2.12 bool btp_parse_string (const char ** *input*, const char * *string*, char ** *result*)

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

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

9.29.2.13 int btp_parse_uint32 (const char ** *input*, uint32_t * *result*)

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

Returns

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

9.29.2.14 void* btp_realloc (void * *ptr*, size_t *size*)

Never returns NULL.

9.29.2.15 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.

9.29.2.16 int btp_skip_char_cspan (const char ** *input*, const char * *reject*)

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

Returns

The number of characters skipped.

9.29.2.17 `bool btp_skip_char_limited (const char ** input, const char * allowed)`

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

9.29.2.18 `int btp_skip_char_sequence (const char ** input, char c)`

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

9.29.2.19 `int btp_skip_char_span (const char ** input, const char * chars)`

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

Returns

The number of characters skipped.

9.29.2.20 `int btp_skip_char_span_location (const char ** input, const char * chars, int * line, int * column)`

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

Parameters

<i>line</i>	Starts from 1. Corresponds to the returned number.
<i>column</i>	Starts from 0. Corresponds to the returned number.

Returns

The number of characters skipped.

9.29.2.21 `int btp_skip_hexadecimal_0xuint (const char ** input)`

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

Returns

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

9.29.2.22 `int btp_skip_hexadecimal_uint (const char ** input)`

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

Returns

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

9.29.2.23 `int btp_skip_string (const char ** input, const char * string)`

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

Returns

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

9.29.2.24 `int btp_skip_uint (const char ** input)`

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

Returns

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

9.29.2.25 `char* btp_strchr_location (const char * s, int c, int * line, int * column)`

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

Parameters

<i>line</i>	Starts from 1. Its value is valid only when this function does not return NULL.
<i>column</i>	Starts from 0. Its value is valid only when this function does not return NULL.

9.29.2.26 `int btp_strcmp0 (const char * s1, const char * s2)`

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

9.29.2.27 `char* btp_strdup (const char * s)`

Never returns NULL.

9.29.2.28 `char* btp_strndup (const char * s, size_t n)`

Never returns NULL.

9.29.2.29 `size_t btp_strspn_location (const char * s, const char * accept, int * line, int * column)`

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

Parameters

<i>line</i>	Starts from 1.
<i>column</i>	Starts from 0.

9.29.2.30 `char* btp_strstr_location (const char * haystack, const char * needle, int * line, int * column)`

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

Parameters

<i>line</i>	Starts from 1. Its value is valid only when this function does not return NULL.
<i>column</i>	Starts from 0. Its value is valid only when this function does not return NULL.

9.29.2.31 `char* btp_vasprintf (const char * format, va_list p)`

Never returns NULL.

9.29.3 Variable Documentation

9.29.3.1 `bool btp_debug_parser`

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

Chapter 10

Example Documentation

10.1 `/home/karel/devel/btparser/lib/koops_frame.h`

Timestamp may be present in the oops lines.

[123456.654321] [65.470000]

Chapter 11

Known Bugs

Empty.

Chapter 12

Wishlist

Stack trace for kerneloopses, Python, and Java.

Index

address
 btp_core_frame, 29
 btp_elf_plt_entry, 35
 btp_gdb_frame, 36
 btp_koops_frame, 46

alloc
 btp_strbuf, 53

btp_asprintf
 utils.h, 123

btp_bin2hex
 utils.h, 123

btp_callgraph, 27
 callees, 28

btp_callgraph_extend
 callgraph.h, 56

btp_cluster, 28

btp_cluster_free
 cluster.h, 57

btp_cluster_new
 cluster.h, 57

btp_core_frame, 28
 address, 29
 build_id, 29
 fingerprint, 29
 next, 29

btp_core_frame_append
 core_frame.h, 59

btp_core_frame_cmp
 core_frame.h, 60

btp_core_frame_dup
 core_frame.h, 60

btp_core_frame_free
 core_frame.h, 60

btp_core_frame_init
 core_frame.h, 60

btp_core_frame_new
 core_frame.h, 60

btp_core_frame_to_json
 core_frame.h, 61

btp_core_stacktrace, 30
 crash_thread, 30
 signal, 30

btp_core_stacktrace_dup
 core_stacktrace.h, 62

btp_core_stacktrace_free
 core_stacktrace.h, 62

btp_core_stacktrace_get_thread_count
 core_stacktrace.h, 62

btp_core_stacktrace_init
 core_stacktrace.h, 62

btp_core_stacktrace_new
 core_stacktrace.h, 62

btp_core_stacktrace_parse
 core_stacktrace.h, 63

btp_core_stacktrace_to_json
 core_stacktrace.h, 63

btp_core_thread, 31
 frames, 31
 next, 31

btp_core_thread_append
 core_thread.h, 64

btp_core_thread_cmp
 core_thread.h, 64

btp_core_thread_dup
 core_thread.h, 64

btp_core_thread_free
 core_thread.h, 64

btp_core_thread_get_frame_count
 core_thread.h, 65

btp_core_thread_init
 core_thread.h, 65

btp_core_thread_new
 core_thread.h, 65

btp_deb_package, 32

btp_debug_parser
 utils.h, 128

btp_dendrogram, 32
 merge_levels, 32

btp_dendrogram_cut
 cluster.h, 57

btp_dendrogram_free
 cluster.h, 57

btp_dendrogram_new
 cluster.h, 57

btp_disasm_get_function_instructions
 disasm.h, 67

btp_dist_thread_type
 metrics.h, 111

- btp_distances, 33
- btp_distances_cluster_objects
 - cluster.h, 58
- btp_distances_dup
 - metrics.h, 111
- btp_distances_free
 - metrics.h, 111
- btp_distances_get_distance
 - metrics.h, 111
- btp_distances_new
 - metrics.h, 111
- btp_distances_set_distance
 - metrics.h, 112
- btp_elf_fde, 33
 - length, 34
 - start_address, 34
- btp_elf_get_eh_frame
 - elves.h, 68
- btp_elf_get_procedure_linkage_table
 - elves.h, 68
- btp_elf_plt_entry, 34
 - address, 35
 - symbol_name, 35
- btp_file_to_string
 - utils.h, 124
- btp_gdb_frame, 35
 - address, 36
 - function_name, 36
 - function_type, 36
 - library_name, 36
 - next, 36
 - number, 36
 - signal_handler_called, 36
 - source_file, 36
 - source_line, 36
- btp_gdb_frame_append
 - gdb_frame.h, 70
- btp_gdb_frame_append_to_str
 - gdb_frame.h, 70
- btp_gdb_frame_calls_func
 - gdb_frame.h, 70
- btp_gdb_frame_cmp
 - gdb_frame.h, 70
- btp_gdb_frame_cmp_simple
 - gdb_frame.h, 71
- btp_gdb_frame_dup
 - gdb_frame.h, 71
- btp_gdb_frame_free
 - gdb_frame.h, 71
- btp_gdb_frame_init
 - gdb_frame.h, 72
- btp_gdb_frame_new
 - gdb_frame.h, 72
- btp_gdb_frame_parse
 - gdb_frame.h, 72
- btp_gdb_frame_parse_address_in_function
 - gdb_frame.h, 72
- btp_gdb_frame_parse_file_location
 - gdb_frame.h, 73
- btp_gdb_frame_parse_frame_start
 - gdb_frame.h, 73
- btp_gdb_frame_parse_function_call
 - gdb_frame.h, 73
- btp_gdb_frame_parse_function_name
 - gdb_frame.h, 73
- btp_gdb_frame_parse_function_name_braces
 - gdb_frame.h, 74
- btp_gdb_frame_parse_function_name_chunk
 - gdb_frame.h, 74
- btp_gdb_frame_parse_function_name_template
 - gdb_frame.h, 74
- btp_gdb_frame_parse_header
 - gdb_frame.h, 75
- btp_gdb_frame_parseadd_operator
 - gdb_frame.h, 75
- btp_gdb_frame_remove_func_prefix
 - gdb_frame.h, 75
- btp_gdb_frame_skip_function_args
 - gdb_frame.h, 75
- btp_gdb_normalize_optimize_thread
 - normalize.h, 113
- btp_gdb_sharedlib, 37
- btp_gdb_sharedlib_append
 - gdb_sharedlib.h, 77
- btp_gdb_sharedlib_count
 - gdb_sharedlib.h, 77
- btp_gdb_sharedlib_dup
 - gdb_sharedlib.h, 77
- btp_gdb_sharedlib_find_address
 - gdb_sharedlib.h, 77
- btp_gdb_sharedlib_free
 - gdb_sharedlib.h, 77
- btp_gdb_sharedlib_init
 - gdb_sharedlib.h, 78
- btp_gdb_sharedlib_new
 - gdb_sharedlib.h, 78
- btp_gdb_sharedlib_parse
 - gdb_sharedlib.h, 78
- btp_gdb_stacktrace, 37
 - crash, 38
 - libs, 38
- btp_gdb_stacktrace_dup
 - gdb_stacktrace.h, 79
- btp_gdb_stacktrace_find_crash_thread
 - gdb_stacktrace.h, 79
- btp_gdb_stacktrace_free
 - gdb_stacktrace.h, 80
- btp_gdb_stacktrace_get_crash_frame

- gdb_stacktrace.h, 80
- btg_gdb_stacktrace_get_duplication_hash
 - gdb_stacktrace.h, 80
- btg_gdb_stacktrace_get_optimized_thread
 - gdb_stacktrace.h, 81
- btg_gdb_stacktrace_get_thread_count
 - gdb_stacktrace.h, 81
- btg_gdb_stacktrace_init
 - gdb_stacktrace.h, 81
- btg_gdb_stacktrace_limit_frame_depth
 - gdb_stacktrace.h, 81
- btg_gdb_stacktrace_new
 - gdb_stacktrace.h, 81
- btg_gdb_stacktrace_parse
 - gdb_stacktrace.h, 82
- btg_gdb_stacktrace_parse_header
 - gdb_stacktrace.h, 82
- btg_gdb_stacktrace_quality_complex
 - gdb_stacktrace.h, 83
- btg_gdb_stacktrace_quality_simple
 - gdb_stacktrace.h, 83
- btg_gdb_stacktrace_remove_threads_except_one
 - gdb_stacktrace.h, 83
- btg_gdb_stacktrace_set_libnames
 - gdb_stacktrace.h, 84
- btg_gdb_stacktrace_to_text
 - gdb_stacktrace.h, 84
- btg_gdb_thread, 39
 - frames, 39
 - next, 39
- btg_gdb_thread_append
 - gdb_thread.h, 85
- btg_gdb_thread_append_to_str
 - gdb_thread.h, 85
- btg_gdb_thread_cmp
 - gdb_thread.h, 85
- btg_gdb_thread_dup
 - gdb_thread.h, 85
- btg_gdb_thread_format_funs
 - gdb_thread.h, 86
- btg_gdb_thread_free
 - gdb_thread.h, 86
- btg_gdb_thread_get_frame_count
 - gdb_thread.h, 86
- btg_gdb_thread_init
 - gdb_thread.h, 86
- btg_gdb_thread_new
 - gdb_thread.h, 86
- btg_gdb_thread_parse
 - gdb_thread.h, 86
- btg_gdb_thread_parse_funs
 - gdb_thread.h, 87
- btg_gdb_thread_quality
 - gdb_thread.h, 87
- btg_gdb_thread_quality_counts
 - gdb_thread.h, 87
- btg_gdb_thread_remove_frame
 - gdb_thread.h, 88
- btg_gdb_thread_remove_frames_above
 - gdb_thread.h, 88
- btg_gdb_thread_remove_frames_below_n
 - gdb_thread.h, 88
- btg_gdb_thread_skip_lwp
 - gdb_thread.h, 88
- btg_gdb_threads_compare
 - metrics.h, 112
- btg_glibc_thread_find_exit_frame
 - normalize.h, 113
- btg_java_exception, 40
 - frames, 40
 - inner, 40
 - message, 40
 - name, 40
- btg_java_exception_append_to_str
 - java_exception.h, 90
- btg_java_exception_cmp
 - java_exception.h, 90
- btg_java_exception_dup
 - java_exception.h, 90
- btg_java_exception_free
 - java_exception.h, 90
- btg_java_exception_get_frame_count
 - java_exception.h, 90
- btg_java_exception_init
 - java_exception.h, 90
- btg_java_exception_new
 - java_exception.h, 91
- btg_java_exception_parse
 - java_exception.h, 91
- btg_java_exception_pop
 - java_exception.h, 91
- btg_java_exception_quality
 - java_exception.h, 91
- btg_java_exception_quality_counts
 - java_exception.h, 92
- btg_java_exception_remove_frame
 - java_exception.h, 92
- btg_java_exception_remove_frames_above
 - java_exception.h, 92
- btg_java_exception_remove_frames_below_n
 - java_exception.h, 93
- btg_java_frame, 41
 - class_path, 41
 - file_line, 41
 - file_name, 41
 - function_name, 41
 - is_native, 41
- btg_java_frame_append_to_str

- java_frame.h, 94
- btp_java_frame_cmp
 - java_frame.h, 94
- btp_java_frame_dup
 - java_frame.h, 94
- btp_java_frame_free
 - java_frame.h, 95
- btp_java_frame_init
 - java_frame.h, 95
- btp_java_frame_new
 - java_frame.h, 95
- btp_java_frame_parse
 - java_frame.h, 95
- btp_java_stacktrace, 42
 - threads, 42
- btp_java_stacktrace_cmp
 - java_stacktrace.h, 96
- btp_java_stacktrace_dup
 - java_stacktrace.h, 96
- btp_java_stacktrace_free
 - java_stacktrace.h, 97
- btp_java_stacktrace_init
 - java_stacktrace.h, 97
- btp_java_stacktrace_new
 - java_stacktrace.h, 97
- btp_java_stacktrace_parse
 - java_stacktrace.h, 97
- btp_java_thread, 43
 - exception, 43
 - name, 43
 - next, 44
- btp_java_thread_append
 - java_thread.h, 99
- btp_java_thread_append_to_str
 - java_thread.h, 99
- btp_java_thread_cmp
 - java_thread.h, 99
- btp_java_thread_dup
 - java_thread.h, 99
- btp_java_thread_format_funs
 - java_thread.h, 99
- btp_java_thread_free
 - java_thread.h, 100
- btp_java_thread_get_frame_count
 - java_thread.h, 100
- btp_java_thread_init
 - java_thread.h, 100
- btp_java_thread_new
 - java_thread.h, 100
- btp_java_thread_parse
 - java_thread.h, 100
- btp_java_thread_parse_funs
 - java_thread.h, 100
- btp_java_thread_quality
 - java_thread.h, 101
- btp_java_thread_quality_counts
 - java_thread.h, 101
- btp_java_thread_remove_frame
 - java_thread.h, 101
- btp_java_thread_remove_frames_above
 - java_thread.h, 101
- btp_java_thread_remove_frames_below_n
 - java_thread.h, 102
- btp_json_settings, 44
- btp_json_value, 44
- btp_koops_frame, 45
 - address, 46
 - from_address, 46
 - from_function_name, 46
 - from_module_name, 46
 - function_name, 46
 - module_name, 46
 - reliable, 46
- btp_koops_frame_append
 - koops_frame.h, 103
- btp_koops_frame_cmp
 - koops_frame.h, 103
- btp_koops_frame_dup
 - koops_frame.h, 103
- btp_koops_frame_free
 - koops_frame.h, 104
- btp_koops_frame_init
 - koops_frame.h, 104
- btp_koops_frame_new
 - koops_frame.h, 104
- btp_koops_frame_to_json
 - koops_frame.h, 104
- btp_koops_stacktrace, 47
 - modules, 47
 - taint_mce, 47
 - taint_module_proprietary, 48
 - taint_page_release, 48
- btp_koops_stacktrace_dup
 - koops_stacktrace.h, 105
- btp_koops_stacktrace_free
 - koops_stacktrace.h, 106
- btp_koops_stacktrace_get_frame_count
 - koops_stacktrace.h, 106
- btp_koops_stacktrace_init
 - koops_stacktrace.h, 106
- btp_koops_stacktrace_new
 - koops_stacktrace.h, 106
- btp_koops_stacktrace_parse
 - koops_stacktrace.h, 106
- btp_koops_stacktrace_remove_frame
 - koops_stacktrace.h, 106
- btp_location, 48
 - column, 48

- line, 48
- message, 48
- btp_location_add
 - location.h, 108
- btp_location_add_ext
 - location.h, 108
- btp_location_cmp
 - location.h, 108
- btp_location_eat_char
 - location.h, 109
- btp_location_eat_char_ext
 - location.h, 109
- btp_location_init
 - location.h, 109
- btp_location_to_string
 - location.h, 109
- btp_malloc
 - utils.h, 124
- btp_mallocz
 - utils.h, 124
- btp_normalize_gdb_paired_unknown_function_names
 - normalize.h, 113
- btp_operating_system, 49
- btp_parse_char_cspan
 - utils.h, 124
- btp_parse_char_limited
 - utils.h, 124
- btp_parse_char_span
 - utils.h, 124
- btp_parse_digit
 - utils.h, 124
- btp_parse_hexadecimal_0xuint64
 - utils.h, 124
- btp_parse_hexadecimal_uint64
 - utils.h, 125
- btp_parse_string
 - utils.h, 125
- btp_parse_uint32
 - utils.h, 125
- btp_python_frame, 49
- btp_python_frame_dup
 - python_frame.h, 114
- btp_python_frame_free
 - python_frame.h, 114
- btp_python_frame_init
 - python_frame.h, 115
- btp_python_frame_new
 - python_frame.h, 115
- btp_python_stacktrace, 50
- btp_python_stacktrace_dup
 - python_stacktrace.h, 116
- btp_python_stacktrace_free
 - python_stacktrace.h, 116
- btp_python_stacktrace_init
 - python_stacktrace.h, 116
- btp_python_stacktrace_new
 - python_stacktrace.h, 116
- btp_realloc
 - utils.h, 125
- btp_report, 50
- btp_rpm_consistency, 51
- btp_rpm_package, 52
- btp_rpm_package_append
 - rpm.h, 117
- btp_sha1_state, 52
- btp_skip_char
 - utils.h, 125
- btp_skip_char_cspan
 - utils.h, 125
- btp_skip_char_limited
 - utils.h, 125
- btp_skip_char_sequence
 - utils.h, 126
- btp_skip_char_span
 - utils.h, 126
- btp_skip_char_span_location
 - utils.h, 126
- btp_skip_hexadecimal_0xuint
 - utils.h, 126
- btp_skip_hexadecimal_uint
 - utils.h, 126
- btp_skip_string
 - utils.h, 126
- btp_skip_uint
 - utils.h, 127
- btp_strbuf, 53
 - alloc, 53
 - len, 53
- btp_strbuf_append_char
 - strbuf.h, 120
- btp_strbuf_append_str
 - strbuf.h, 120
- btp_strbuf_append_strf
 - strbuf.h, 120
- btp_strbuf_append_strfv
 - strbuf.h, 120
- btp_strbuf_clear
 - strbuf.h, 120
- btp_strbuf_free
 - strbuf.h, 120
- btp_strbuf_free_nobuf
 - strbuf.h, 120
- btp_strbuf_grow
 - strbuf.h, 120
- btp_strbuf_init
 - strbuf.h, 120
- btp_strbuf_new
 - strbuf.h, 120

- btp_strbuf_prepend_str
 - strbuf.h, 121
- btp_strbuf_prepend_strf
 - strbuf.h, 121
- btp_strbuf_prepend_strfv
 - strbuf.h, 121
- btp_strchr_location
 - utils.h, 127
- btp_strcmp0
 - utils.h, 127
- btp_strdup
 - utils.h, 127
- btp_strndup
 - utils.h, 127
- btp_strspn_location
 - utils.h, 127
- btp_strstr_location
 - utils.h, 127
- btp_unstrip_entry, 53
- btp_vasprintf
 - utils.h, 128
- build_id
 - btp_core_frame, 29
- callees
 - btp_callgraph, 28
- callgraph.h, 55
 - btp_callgraph_extend, 56
- class_path
 - btp_java_frame, 41
- cluster.h, 56
 - btp_cluster_free, 57
 - btp_cluster_new, 57
 - btp_dendrogram_cut, 57
 - btp_dendrogram_free, 57
 - btp_dendrogram_new, 57
 - btp_distances_cluster_objects, 58
- column
 - btp_location, 48
- core_fingerprint.h, 58
- core_frame.h, 59
 - btp_core_frame_append, 59
 - btp_core_frame_cmp, 60
 - btp_core_frame_dup, 60
 - btp_core_frame_free, 60
 - btp_core_frame_init, 60
 - btp_core_frame_new, 60
 - btp_core_frame_to_json, 61
- core_stacktrace.h, 61
 - btp_core_stacktrace_dup, 62
 - btp_core_stacktrace_free, 62
 - btp_core_stacktrace_get_thread_count, 62
 - btp_core_stacktrace_init, 62
 - btp_core_stacktrace_new, 62
 - btp_core_stacktrace_parse, 63
 - btp_core_stacktrace_to_json, 63
- core_thread.h, 63
 - btp_core_thread_append, 64
 - btp_core_thread_cmp, 64
 - btp_core_thread_dup, 64
 - btp_core_thread_free, 64
 - btp_core_thread_get_frame_count, 65
 - btp_core_thread_init, 65
 - btp_core_thread_new, 65
- crash
 - btp_gdb_stacktrace, 38
- crash_thread
 - btp_core_stacktrace, 30
- deb.h, 65
- disasm.h, 66
 - btp_disasm_get_function_instructions, 67
- elves.h, 67
 - btp_elf_get_eh_frame, 68
 - btp_elf_get_procedure_linkage_table, 68
- exception
 - btp_java_thread, 43
- file_line
 - btp_java_frame, 41
- file_name
 - btp_java_frame, 41
- fingerprint
 - btp_core_frame, 29
- frames
 - btp_core_thread, 31
 - btp_gdb_thread, 39
 - btp_java_exception, 40
- from_address
 - btp_koops_frame, 46
- from_function_name
 - btp_koops_frame, 46
- from_module_name
 - btp_koops_frame, 46
- function_name
 - btp_gdb_frame, 36
 - btp_java_frame, 41
 - btp_koops_frame, 46
- function_type
 - btp_gdb_frame, 36
- gdb_frame.h, 68
 - btp_gdb_frame_append, 70
 - btp_gdb_frame_append_to_str, 70
 - btp_gdb_frame_calls_func, 70
 - btp_gdb_frame_cmp, 70
 - btp_gdb_frame_cmp_simple, 71

- btg_gdb_frame_dup, 71
 - btg_gdb_frame_free, 71
 - btg_gdb_frame_init, 72
 - btg_gdb_frame_new, 72
 - btg_gdb_frame_parse, 72
 - btg_gdb_frame_parse_address_in_function, 72
 - btg_gdb_frame_parse_file_location, 73
 - btg_gdb_frame_parse_frame_start, 73
 - btg_gdb_frame_parse_function_call, 73
 - btg_gdb_frame_parse_function_name, 73
 - btg_gdb_frame_parse_function_name_braces, 74
 - btg_gdb_frame_parse_function_name_chunk, 74
 - btg_gdb_frame_parse_function_name_template, 74
 - btg_gdb_frame_parse_header, 75
 - btg_gdb_frame_parseadd_operator, 75
 - btg_gdb_frame_remove_func_prefix, 75
 - btg_gdb_frame_skip_function_args, 75
- gdb_sharedlib.h, 76
 - btg_gdb_sharedlib_append, 77
 - btg_gdb_sharedlib_count, 77
 - btg_gdb_sharedlib_dup, 77
 - btg_gdb_sharedlib_find_address, 77
 - btg_gdb_sharedlib_free, 77
 - btg_gdb_sharedlib_init, 78
 - btg_gdb_sharedlib_new, 78
 - btg_gdb_sharedlib_parse, 78
- gdb_stacktrace.h, 78
 - btg_gdb_stacktrace_dup, 79
 - btg_gdb_stacktrace_find_crash_thread, 79
 - btg_gdb_stacktrace_free, 80
 - btg_gdb_stacktrace_get_crash_frame, 80
 - btg_gdb_stacktrace_get_duplication_hash, 80
 - btg_gdb_stacktrace_get_optimized_thread, 81
 - btg_gdb_stacktrace_get_thread_count, 81
 - btg_gdb_stacktrace_init, 81
 - btg_gdb_stacktrace_limit_frame_depth, 81
 - btg_gdb_stacktrace_new, 81
 - btg_gdb_stacktrace_parse, 82
 - btg_gdb_stacktrace_parse_header, 82
 - btg_gdb_stacktrace_quality_complex, 83
 - btg_gdb_stacktrace_quality_simple, 83
 - btg_gdb_stacktrace_remove_threads_except_one, 83
 - btg_gdb_stacktrace_set_libnames, 84
 - btg_gdb_stacktrace_to_text, 84
- gdb_thread.h, 84
 - btg_gdb_thread_append, 85
 - btg_gdb_thread_append_to_str, 85
 - btg_gdb_thread_cmp, 85
 - btg_gdb_thread_dup, 85
 - btg_gdb_thread_format_funs, 86
 - btg_gdb_thread_free, 86
 - btg_gdb_thread_get_frame_count, 86
 - btg_gdb_thread_init, 86
 - btg_gdb_thread_new, 86
 - btg_gdb_thread_parse, 86
 - btg_gdb_thread_parse_funs, 87
 - btg_gdb_thread_quality, 87
 - btg_gdb_thread_quality_counts, 87
 - btg_gdb_thread_remove_frame, 88
 - btg_gdb_thread_remove_frames_above, 88
 - btg_gdb_thread_remove_frames_below_n, 88
 - btg_gdb_thread_skip_lwp, 88
- btg_java_exception, 40
 - is_native
 - btg_java_frame, 41
- java_exception.h, 88
 - btg_java_exception_append_to_str, 90
 - btg_java_exception_cmp, 90
 - btg_java_exception_dup, 90
 - btg_java_exception_free, 90
 - btg_java_exception_get_frame_count, 90
 - btg_java_exception_init, 90
 - btg_java_exception_new, 91
 - btg_java_exception_parse, 91
 - btg_java_exception_pop, 91
 - btg_java_exception_quality, 91
 - btg_java_exception_quality_counts, 92
 - btg_java_exception_remove_frame, 92
 - btg_java_exception_remove_frames_above, 92
 - btg_java_exception_remove_frames_below_n, 93
- java_frame.h, 93
 - btg_java_frame_append_to_str, 94
 - btg_java_frame_cmp, 94
 - btg_java_frame_dup, 94
 - btg_java_frame_free, 95
 - btg_java_frame_init, 95
 - btg_java_frame_new, 95
 - btg_java_frame_parse, 95
- java_stacktrace.h, 96
 - btg_java_stacktrace_cmp, 96
 - btg_java_stacktrace_dup, 96
 - btg_java_stacktrace_free, 97
 - btg_java_stacktrace_init, 97
 - btg_java_stacktrace_new, 97
 - btg_java_stacktrace_parse, 97
- java_thread.h, 98
 - btg_java_thread_append, 99
 - btg_java_thread_append_to_str, 99
 - btg_java_thread_cmp, 99
 - btg_java_thread_dup, 99
 - btg_java_thread_format_funs, 99
 - btg_java_thread_free, 100
 - btg_java_thread_get_frame_count, 100

- btpt_java_thread_init, 100
- btpt_java_thread_new, 100
- btpt_java_thread_parse, 100
- btpt_java_thread_parse_funs, 100
- btpt_java_thread_quality, 101
- btpt_java_thread_quality_counts, 101
- btpt_java_thread_remove_frame, 101
- btpt_java_thread_remove_frames_above, 101
- btpt_java_thread_remove_frames_below_n, 102
- koops_frame.h, 102
 - btpt_koops_frame_append, 103
 - btpt_koops_frame_cmp, 103
 - btpt_koops_frame_dup, 103
 - btpt_koops_frame_free, 104
 - btpt_koops_frame_init, 104
 - btpt_koops_frame_new, 104
 - btpt_koops_frame_to_json, 104
- koops_stacktrace.h, 104
 - btpt_koops_stacktrace_dup, 105
 - btpt_koops_stacktrace_free, 106
 - btpt_koops_stacktrace_get_frame_count, 106
 - btpt_koops_stacktrace_init, 106
 - btpt_koops_stacktrace_new, 106
 - btpt_koops_stacktrace_parse, 106
 - btpt_koops_stacktrace_remove_frame, 106
- len
 - btpt_strbuf, 53
- length
 - btpt_elf_fde, 34
- library_name
 - btpt_gdb_frame, 36
- libs
 - btpt_gdb_stacktrace, 38
- line
 - btpt_location, 48
- location.h, 107
 - btpt_location_add, 108
 - btpt_location_add_ext, 108
 - btpt_location_cmp, 108
 - btpt_location_eat_char, 109
 - btpt_location_eat_char_ext, 109
 - btpt_location_init, 109
 - btpt_location_to_string, 109
- merge_levels
 - btpt_dendrogram, 32
- message
 - btpt_java_exception, 40
 - btpt_location, 48
- metrics.h, 109
 - btpt_dist_thread_type, 111
 - btpt_distances_dup, 111
 - btpt_distances_free, 111
 - btpt_distances_get_distance, 111
 - btpt_distances_new, 111
 - btpt_distances_set_distance, 112
 - btpt_gdb_threads_compare, 112
- module_name
 - btpt_koops_frame, 46
- modules
 - btpt_koops_stacktrace, 47
- name
 - btpt_java_exception, 40
 - btpt_java_thread, 43
- next
 - btpt_core_frame, 29
 - btpt_core_thread, 31
 - btpt_gdb_frame, 36
 - btpt_gdb_thread, 39
 - btpt_java_thread, 44
- normalize.h, 112
 - btpt_gdb_normalize_optimize_thread, 113
 - btpt_glibc_thread_find_exit_frame, 113
 - btpt_normalize_gdb_paired_unknown_function_names, 113
- number
 - btpt_gdb_frame, 36
- python_frame.h, 113
 - btpt_python_frame_dup, 114
 - btpt_python_frame_free, 114
 - btpt_python_frame_init, 115
 - btpt_python_frame_new, 115
- python_stacktrace.h, 115
 - btpt_python_stacktrace_dup, 116
 - btpt_python_stacktrace_free, 116
 - btpt_python_stacktrace_init, 116
 - btpt_python_stacktrace_new, 116
- reliable
 - btpt_koops_frame, 46
- rpm.h, 117
 - btpt_rpm_package_append, 117
- sha1.h, 118
- signal
 - btpt_core_stacktrace, 30
- signal_handler_called
 - btpt_gdb_frame, 36
- source_file
 - btpt_gdb_frame, 36
- source_line
 - btpt_gdb_frame, 36
- start_address
 - btpt_elf_fde, 34

- strbuf.h, 119
 - btp_strbuf_append_char, 120
 - btp_strbuf_append_str, 120
 - btp_strbuf_append_strf, 120
 - btp_strbuf_append_strfv, 120
 - btp_strbuf_clear, 120
 - btp_strbuf_free, 120
 - btp_strbuf_free_nobuf, 120
 - btp_strbuf_grow, 120
 - btp_strbuf_init, 120
 - btp_strbuf_new, 120
 - btp_strbuf_prepend_str, 121
 - btp_strbuf_prepend_strf, 121
 - btp_strbuf_prepend_strfv, 121
- symbol_name
 - btp_elf_plt_entry, 35
- taint_mce
 - btp_koops_stacktrace, 47
- taint_module_proprietary
 - btp_koops_stacktrace, 48
- taint_page_release
 - btp_koops_stacktrace, 48
- threads
 - btp_java_stacktrace, 42
- unstrip.h, 121
- utils.h, 122
 - btp_asprintf, 123
 - btp_bin2hex, 123
 - btp_debug_parser, 128
 - btp_file_to_string, 124
 - btp_malloc, 124
 - btp_mallocz, 124
 - btp_parse_char_cspan, 124
 - btp_parse_char_limited, 124
 - btp_parse_char_span, 124
 - btp_parse_digit, 124
 - btp_parse_hexadecimal_0xuint64, 124
 - btp_parse_hexadecimal_uint64, 125
 - btp_parse_string, 125
 - btp_parse_uint32, 125
 - btp_realloc, 125
 - btp_skip_char, 125
 - btp_skip_char_cspan, 125
 - btp_skip_char_limited, 125
 - btp_skip_char_sequence, 126
 - btp_skip_char_span, 126
 - btp_skip_char_span_location, 126
 - btp_skip_hexadecimal_0xuint, 126
 - btp_skip_hexadecimal_uint, 126
 - btp_skip_string, 126
 - btp_skip_uint, 127
 - btp_strchr_location, 127
 - btp_strcmp0, 127
 - btp_strdup, 127
 - btp_strndup, 127
 - btp_strspn_location, 127
 - btp_strstr_location, 127
 - btp_vasprintf, 128