NAME

debuginfod_find_debuginfo − request debuginfo from debuginfod

SYNOPSIS

#include <elfutils/debuginfod.h>

Link with **-ldebuginfod**.

CONNECTION HANDLE

debuginfod_client *debuginfod_begin(void);
void debuginfod_end(debuginfod_client * *client***);**

LOOKUP FUNCTIONS

int debuginfod_find_debuginfo(debuginfod_client * *client***,**
const unsigned char * *build_id***,**
int *build_id_len***,**
char ** *path***);**
int debuginfod_find_executable(debuginfod_client * *client***,**
const unsigned char * *build_id***,**
int *build_id_len***,**
char ** *path***);**
int debuginfod_find_source(debuginfod_client * *client***,**
const unsigned char * *build_id***,**
int *build_id_len***,**
const char * *filename***,**
char ** *path***);**
int debuginfod_find_section(debuginfod_client * *client***,**
const unsigned char * *build_id***,**
int *build_id_len***,**
const char * *section***,**
char ** *path***);**
int debuginfod_find_metadata(debuginfod_client * *client***,**
const char * *key***,**
const char * *value***,**
char ** *path***);**

OPTIONAL FUNCTIONS

typedef int (*debuginfod_progressfn_t)(debuginfod_client * *client***,**
long *a***, long** *b***);**
void debuginfod_set_progressfn(debuginfod_client * *client***,**
debuginfod_progressfn_t *progressfn***);**
int debuginfod_default_progressfn(debuginfod_client * *client***,**
long *a***, long** *b***);**
void debuginfod_set_verbose_fd(debuginfod_client * *client***,**
int *fd***);**
void debuginfod_set_user_data(debuginfod_client * *client***,**
void * *data***);**
void* debuginfod_get_user_data(debuginfod_client * *client***);**
const char* debuginfod_get_url(debuginfod_client * *client***);**
int debuginfod_add_http_header(debuginfod_client * *client***,**
const char* *header***);**

const char* debuginfod_get_headers(debuginfod_client ***client*);**

DESCRIPTION

debuginfod_begin() creates a **debuginfod_client** connection handle that should be used with all other calls. **debuginfod_end**() should be called on the **client** handle to release all state and storage when done.

debuginfod_find_debuginfo(), **debuginfod_find_executable**(), and **debuginfod_find_source**() query the debuginfod server URLs contained in **$DEBUGINFOD_URLS** (see below) for the debuginfo, executable or source file with the given *build_id*. *build_id* should be a pointer to either a null-terminated, lowercase hex string or a binary blob. If *build_id* is given as a hex string, *build_id_len* should be 0. Otherwise *build_id_len* should be the number of bytes in the binary blob.

debuginfod_find_source() also requires a *filename* in order to specify a particular source file. *filename* should be an absolute path that includes the compilation directory of the CU associated with the source file. Relative path names commonly appear in the DWARF file's source directory, but these paths are relative to individual compilation unit AT_comp_dir paths, and yet an executable is made up of multiple CUs. Therefore, to disambiguate, debuginfod expects source queries to prefix relative path names with the CU compilation-directory, followed by a mandatory "/".

Note: the caller may or may not elide **../** or **/../** or extraneous /// sorts of path components in the directory names. debuginfod accepts both forms. Specifically, debuginfod canonicalizes path names according to RFC3986 section 5.2.4 (Remove Dot Segments), plus reducing any // to / in the path.

debuginfod_find_section() queries the debuginfod server URLs contained in **$DEBUGINFOD_URLS** for the binary contents of an ELF/DWARF section contained in a debuginfo or executable file with the given *build_id*. *section* should be the name of the desired ELF/DWARF section. If a server does not support section queries, debuginfod_find_section may query the server for the debuginfo and/or executable with *build_id* in order to retrieve and extract the section.

debuginfod_find_metadata() queries all debuginfod server URLs contained in **$DEBUGINFOD_URLS** for metadata for all matches of a given key/value query against files in their indexes. The resulting file is a JSON document. See the *debuginfod-find(1)* man page for examples of the supported types of key/value queries and their JSON results.

If *path* is not NULL and the query is successful, *path* is set to the path of the file in the cache. The caller must **free**() this value.

The URLs in **$DEBUGINFOD_URLS** may be queried in parallel. As soon as a debuginfod server begins transferring the target file all of the connections to the other servers are closed.

A **client** handle should be used from only one thread at a time. A handle may be reused for a series of lookups, which can improve performance due to retention of connections and caches.

RETURN VALUE

debuginfod_begin returns the **debuginfod_client** handle to use with all other calls. On error **NULL** will be returned and **errno** will be set.

If a find family function is successful, the resulting file is saved to the client cache and a file descriptor to that file is returned. The file descriptor points to the beginning of the file. The caller needs to **close**() this descriptor. Otherwise, a negative error code is returned.

OPTIONAL FUNCTIONS

A small number of optional functions are available to tune or query the operation of the debuginfod client.

PROGRESS CALLBACK

As the **debuginfod_find_***() functions may block for seconds or longer, a progress callback function is called periodically, if configured with **debuginfod_set_progressfn**(). This function sets a new progress callback function (or NULL) for the client handle.

The given callback function is called from the context of each thread that is invoking any of the other lookup functions. It is given two numeric parameters that, if thought of as a numerator *a* and denominator *b*, together represent a completion fraction *a/b*. The denominator may be zero initially, until a quantity such as an exact download size becomes known.

The progress callback function is also the supported way to *interrupt* the download operation. (The library does *not* modify or trigger signals.) The progress callback must return 0 to continue the work, or any other value to stop work as soon as possible. Consequently, the **debuginfod_find_***() function will likely return with an error, but might still succeed.

debuginfod_default_progressfn() is the progress callback function set by the debuginfod client library when the **DEBUGINFOD_PROGRESS** environment variable is set and no progressfn is explicitly set. For more information see **DEBUGINFOD_PROGRESS** below. Custom progress callback functions may call **debuginfod_default_progressfn**() directly to report download status.

VERBOSE OUTPUT

The **debuginfod_find_***() functions may use several techniques to retrieve the requested files, through the cache or through one or multiple servers or file URLs. To show how a query is handled the **debuginfod_set_verbose_fd**() can be used to set a particular file descriptor on which verbose output is given about the query steps and eventual errors encountered.

USER DATA POINTER

A single *void* * pointer associated with the connection handle may be set any time via **debuginfod_set_user_data**()**,** and retrieved via **debuginfod_get_user_data**()**.** The value is undefined if unset.

URL

The URL of the current or most recent outgoing download, if known, may be retrieved via **debuginfod_get_url**() from the progressfn callback, or afterwards. It may be NULL. The resulting string is owned by the library, and must not be modified or freed. The caller should copy it if it is needed beyond the release of the client object.

HTTP HEADER

Before each lookup function is initiated, a client application may add HTTP request headers. These are reset after each lookup function. **deb uginfod_add_http_header**() may be called with strings of the form **"Header: value"**. These strings are copied by the library. A zero return value indicates success, but out-of-memory conditions may result in a non-zero -*ENOMEM*. If the string is in the wrong form -*EINVAL* will be returned.

$DEBUGINFOD_HEADERS_FILE specifies a file to supply headers to outgoing requests. Each non-white-space line of this file is handled as if **debuginfod_add_http_header**() were called on the contents.

Note that the current debuginfod-client library implementation uses libcurl, but you shouldn't rely on that fact. Don't use this function for replacing any standard headers, except for the User-Agent mentioned

below. You can use this function to add authorization information for access control, or to provide optional headers to the server for logging purposes.

By default, the library adds a descriptive *User-Agent:* header to outgoing requests. If the client application adds a header with the same name, this default is suppressed.

During or after a lookup, a client application may call **debuginfod_get_headers**() to gather the subset of HTTP response headers received from the current or most recent debuginfod server. Only those headers prefixed with **X-DEBUGINFOD** (case-insensitive) are kept. They are returned as a single string, with each "header: value" terminated with a \n (not \r\n as in HTTP). It may be NULL. The resulting string is owned by the library, and must not be modified or freed. The caller should copy the returned string if it is needed beyond the release of the client object.

MACROS

DEBUGINFOD_SONAME

Defined to the string that could be passed to **dlopen**(3) if the library is loaded at runtime, for example

```
void *debuginfod_so = dlopen(DEBUGINFOD_SONAME, RTLD_LAZY);
```

SECURITY

If IMA signature(s) are available from the RPMs that contain requested files, then **debuginfod** will extract those signatures into response headers, and **debuginfod_find_***() will perform verification upon the files. Validation policy is controlled via tags inserted into $DEBUGINFOD_URLS. By default, **debuginfod_find_***() acts in ignore mode.

If accessed across HTTP rather than HTTPS, the network should be trustworthy. Authentication information through the internal *libcurl* library is not currently enabled, except for the basic plaintext *http[s]://userid:password@hostname/* style. (The debuginfod server does not perform authentication, but a front-end proxy server could.)

ERRORS

The following list is not comprehensive. Error codes may also originate from calls to various C Library functions.

EACCESS

Denied access to resource located at the URL.

ECONNREFUSED

Unable to connect to remote host. Also returned when an HTTPS connection couldn't be verified (bad certificate).

ECONNRESET

Unable to either send or receive network data.

EHOSTUNREACH

Unable to resolve remote host.

EINVAL

One or more arguments are incorrectly formatted.*b uild_id* may be too long (greater than 256 characters), *filename* may not be an absolute path or a debuginfod URL is malformed.

EIO Unable to write data received from server to local file.

EMLINK
Too many HTTP redirects.

ENETUNREACH
Unable to initialize network connection.

ENOENT
Could not find the resource located at URL. Often this error code indicates that a debuginfod server was successfully contacted but the server could not find the target file.

ENOMEM
System is unable to allocate resources.

ENOSYS
$DEBUGINFOD_URLS is not defined.

ETIME
Query failed due to timeout. **$DEBUGINFOD_TIMEOUT** and **$DEBUGINFOD_MAXTIME** control this.

EF2BIG
Query aborted due to the file requested being too big. The **$DEBUGINFOD_MAXSIZE** controls this.

EBADMSG
File content failed IMA verification against a known signer certificate.

ENOKEY
File content failed IMA verification due to missing signer certificate.

ENODATA
File content failed IMA verification because of a missing signature.

The following information is generic. Some debuginfod clients may internally override defaults listed here.

ENVIRONMENT VARIABLES

$TMPDIR
This environment variable points to a file system to be used for temporary files. The default is /tmp.

$DEBUGINFOD_URLS
This environment variable contains a list of URL prefixes for trusted debuginfod instances. Alternate URL prefixes are separated by space. This environment variable may be set by /etc/profile.d scripts reading /etc/debuginfod/*.urls files.

This environment variable can also contain policy defining tags which dictate the response policy

for verifying per-file IMA signatures in RPMs. As the space seperated list is read left to right, upon encountering a tag, subsequent URLs up to the next tag will be handled using that specified policy. All URLs before the first tag will use the default policy, *ima:ignore*. For example:

```
DEBUGINFOD_URLS="https://foo.com ima:enforcing https://bar.ca http://localh
```

Where foo.com and baz.org use the default *ignore* policy and bar.ca and localhost use an *enforcing* policy. The policy tag may be one of the following:

ima:enforcing Every downloaded file requires a valid signature, fully protecting integrity.

ima:ignore Skips verification altogether, providing no protection.

Alerts of validation failure will be directed as specified in $DEBUGINFOD_VERBOSE.

$DEBUGINFOD_CACHE_PATH

This environment variable governs the location of the cache where downloaded files and cache-control files are kept. The default directory is chosen based on other environment variables, see below.

$DEBUGINFOD_PROGRESS

This environment variable governs the default progress function. If set, and if a progressfn is not explicitly set, then the library will configure a default progressfn. This function will append a simple progress message periodically to stderr. The default is no progress function output.

$DEBUGINFOD_VERBOSE

This environment variable governs the default file descriptor for verbose output. If set, and if a verbose fd is not explicitly set, then the verbose output will be produced on STDERR_FILENO.

$DEBUGINFOD_RETRY_LIMIT

This environment variable governs the default limit of retry attempts. If a query failed with errno other than ENOENT, will initiate several attempts within the limit.

$DEBUGINFOD_TIMEOUT

This environment variable governs the download *commencing* timeout for each debuginfod HTTP connection. A server that fails to provide at least 100K of data within this many seconds is skipped. The default is 90 seconds. (Zero or negative means "no timeout".)

$DEBUGINFOD_MAXTIME

This environment variable dictates how long the client will wait to *complete* the download a file found on a server in seconds. It is best used to ensure that a file is downloaded quickly or be rejected. The default is 0 (infinite time).

$DEBUGINFOD_MAXSIZE

This environment variable dictates the maximum size of a file to download in bytes. This is best used if the user would like to ensure only small files are downloaded. A value of 0 causes no consideration for size, and the client may attempt to download a file of any size. The default is 0 (infinite size).

$DEBUGINFOD_HEADERS_FILE

This environment variable points to a file that supplies headers to outbound HTTP requests, one per line. The header lines shouldn't end with CRLF, unless that's the system newline convention. Whitespace-only lines are skipped.

$DEBUGINFOD_IMA_CERT_PATH
This environment variable contains a list of absolute directory paths holding X.509 certificates for RPM per-file IMA-verification. Alternate paths are separated by colons.

CACHE

Before each query, the debuginfod client library checks for a need to clean the cache. If it's time to clean, the library traverses the cache directory and removes downloaded debuginfo-related artifacts and newly empty directories, if they have not been accessed recently.

Control files are located directly under the cache directory. They contain simple decimal numbers to set cache-related configuration parameters. If the files do not exist, the client library creates the files with the default parameter values as content.

After each query, the debuginfod client library deposits newly received files into a directory & file that is named based on the build-id. A failed query is also cached by a special file. The naming convention used for these artifacts is deliberately **undocumented**.

$XDG_CACHE_HOME/debuginfod_client/
Default cache directory, if $XDG_CACHE_HOME is set.

$HOME/.cache/debuginfod_client/
Default cache directory, if $XDG_CACHE_HOME is not set.

$HOME/.debuginfod_client_cache/
Deprecated cache directory, used only if preexisting.

cache_clean_interval_s
This control file gives the interval between cache cleaning rounds, in seconds. The default is 86400, one day. 0 means "immediately".

max_unused_age_s
This control file sets how long unaccessed debuginfo-related files are retained, in seconds. The default is 604800, one week. 0 means "immediately".

cache_miss_s
This control file sets how long to remember a query failure, in seconds. New queries for the same artifacts within this time window are short-circuited (returning an immediate failure instead of sending a new query to servers). This accelerates queries that probably would still fail. The default is 600, 10 minutes. 0 means "forget immediately".

metadata_retention_s
This control file sets how long to remember the results of a metadata query. New queries for the same artifacts within this time window are short-circuited (repeating the same results). This accelerates queries that probably would probably have the same results. The default is 3600, 1 hour. 0 means "do not retain".

SEE ALSO

debuginfod(8)