

**NAME**

**libbe** - library for creating, destroying and modifying ZFS boot environments

**LIBRARY**

Boot Environment Library (libbe, -lbe)

**SYNOPSIS**

**#include** <be.h>

*libbe\_handle\_t \*hdl*

**libbe\_init**(*const char \*be\_root*);

*void*

**libbe\_close**(*libbe\_handle\_t \*hdl*);

*const char \**

**be\_active\_name**(*libbe\_handle\_t \*hdl*);

*const char \**

**be\_active\_path**(*libbe\_handle\_t \*hdl*);

*const char \**

**be\_nextboot\_name**(*libbe\_handle\_t \*hdl*);

*const char \**

**be\_nextboot\_path**(*libbe\_handle\_t \*hdl*);

*const char \**

**be\_root\_path**(*libbe\_handle\_t \*hdl*);

*int*

**be\_snapshot**(*libbe\_handle\_t \*hdl, const char \*be\_name, const char \*snap\_name, bool recursive, char \*result*);

*bool*

**be\_is\_auto\_snapshot\_name**(*libbe\_handle\_t \*hdl, const char \*snap*);

*int*

**be\_create**(*libbe\_handle\_t \*hdl, const char \*be\_name*);

*int*

**be\_create\_depth**(*libbe\_handle\_t* \*hdl, *const char* \*be\_name, *const char* \*snap, *int* depth);

*int*

**be\_create\_from\_existing**(*libbe\_handle\_t* \*hdl, *const char* \*be\_name, *const char* \*be\_origin);

*int*

**be\_create\_from\_existing\_snap**(*libbe\_handle\_t* \*hdl, *const char* \*be\_name, *const char* \*snap);

*int*

**be\_rename**(*libbe\_handle\_t* \*hdl, *const char* \*be\_old, *const char* \*be\_new);

*int*

**be\_activate**(*libbe\_handle\_t* \*hdl, *const char* \*be\_name, *bool* temporary);

*int*

**be\_deactivate**(*libbe\_handle\_t* \*hdl, *const char* \*be\_name, *bool* temporary);

*int*

**be\_destroy**(*libbe\_handle\_t* \*hdl, *const char* \*be\_name, *int* options);

*void*

**be\_nicenum**(*uint64\_t* num, *char* \*buf, *size\_t* bufsz);

*int*

**be\_mount**(*libbe\_handle\_t* \*hdl, *const char* \*be\_name, *const char* \*mntpoint, *int* flags, *char* \*result);

*int*

**be\_mounted\_at**(*libbe\_handle\_t* \*hdl, *const char* \*path, *nvlist\_t* \*details);

*int*

**be\_unmount**(*libbe\_handle\_t* \*hdl, *const char* \*be\_name, *int* flags);

*int*

**libbe\_errno**(*libbe\_handle\_t* \*hdl);

*const char* \*

**libbe\_error\_description**(*libbe\_handle\_t* \*hdl);

*void*

**libbe\_print\_on\_error**(*libbe\_handle\_t \*hdl, bool doprint*);

*int*

**be\_root\_concat**(*libbe\_handle\_t \*hdl, const char \*be\_name, char \*result*);

*int*

**be\_validate\_name**(*libbe\_handle\_t \*hdl, const char \*be\_name*);

*int*

**be\_validate\_snap**(*libbe\_handle\_t \*hdl, const char \*snap*);

*int*

**be\_exists**(*libbe\_handle\_t \*hdl, const char \*be\_name*);

*int*

**be\_export**(*libbe\_handle\_t \*hdl, const char \*be\_name, int fd*);

*int*

**be\_import**(*libbe\_handle\_t \*hdl, const char \*be\_name, int fd*);

*int*

**be\_prop\_list\_alloc**(*nvlist\_t \*\*prop\_list*);

*int*

**be\_get\_bootenv\_props**(*libbe\_handle\_t \*hdl, nvlist\_t \*be\_list*);

*int*

**be\_get\_dataset\_props**(*libbe\_handle\_t \*hdl, const char \*ds\_name, nvlist\_t \*props*);

*int*

**be\_get\_dataset\_snapshots**(*libbe\_handle\_t \*hdl, const char \*ds\_name, nvlist\_t \*snap\_list*);

*void*

**be\_prop\_list\_free**(*nvlist\_t \*prop\_list*);

## DESCRIPTION

**libbe** interfaces with **libzfs** to provide a set of functions for various operations regarding ZFS boot environments including "deep" boot environments in which a boot environments has child datasets.

A context structure is passed to each function, allowing for a small amount of state to be retained, such

as errors from previous operations. **libbe** may be configured to print the corresponding error message to stderr when an error is encountered with **libbe\_print\_on\_error()**.

All functions returning an *int* return 0 on success, or a **libbe** errno otherwise as described in *DIAGNOSTICS*.

The **libbe\_init()** function takes an optional BE root and initializes **libbe**, returning a *libbe\_handle\_t* \* on success, or NULL on error. If a BE root is supplied, **libbe** will only operate out of that pool and BE root. An error may occur if:

- /boot and / are not on the same filesystem and device,
- libzfs fails to initialize,
- The system has not been properly booted with a ZFS boot environment,
- libbe** fails to open the zpool the active boot environment resides on, or
- libbe** fails to locate the boot environment that is currently mounted.

The **libbe\_close()** function frees all resources previously acquired in **libbe\_init()**, invalidating the handle in the process.

The **be\_active\_name()** function returns the name of the currently booted boot environment. This boot environment may not belong to the same BE root as the root libbe is operating on!

The **be\_active\_path()** function returns the full path of the currently booted boot environment. This boot environment may not belong to the same BE root as the root libbe is operating on!

The **be\_nextboot\_name()** function returns the name of the boot environment that will be active on reboot.

The **be\_nextboot\_path()** function returns the full path of the boot environment that will be active on reboot.

The **be\_root\_path()** function returns the boot environment root path.

The **be\_snapshot()** function creates a snapshot of *be\_name* named *snap\_name*. A NULL *snap\_name* may be used, indicating that **be\_snapshot()** should derive the snapshot name from the current date and time. If *recursive* is set, then **be\_snapshot()** will recursively snapshot the dataset. If *result* is not NULL, then it will be populated with the final "*be\_name@snap\_name*".

The **be\_is\_auto\_snapshot\_name()** function is used to determine if the given snapshot name matches the format that the **be\_snapshot()** function will use by default if it is not given a snapshot name to use. It

returns true if the name matches the format, and false if it does not.

The **be\_create()** function creates a boot environment with the given name. The new boot environment will be created from a recursive snapshot of the currently booted boot environment.

The **be\_create\_depth()** function creates a boot environment with the given name from an existing snapshot. The depth parameter specifies the depth of recursion that will be cloned from the existing snapshot. A depth of '0' is no recursion and '-1' is unlimited (i.e., a recursive boot environment).

The **be\_create\_from\_existing()** function creates a boot environment with the given name from the name of an existing boot environment. A recursive snapshot will be made of the origin boot environment, and the new boot environment will be created from that.

The **be\_create\_from\_existing\_snap()** function creates a recursive boot environment with the given name from an existing snapshot.

The **be\_rename()** function renames a boot environment without unmounting it, as if renamed with the **-u** argument were passed to **zfs rename**

The **be\_activate()** function makes a boot environment active on the next boot. If the *temporary* flag is set, then it will be active for the next boot only, as done by **zfsbootcfg(8)**.

The **be\_deactivate()** function deactivates a boot environment. If the *temporary* flag is set, then it will cause removal of boot once configuration, set by **be\_activate()** function or by **zfsbootcfg(8)**. If the *temporary* flag is not set, **be\_deactivate()** function will set **zfs canmount** property to **noauto**.

The **be\_destroy()** function will recursively destroy the given boot environment. It will not destroy a mounted boot environment unless the **BE\_DESTROY\_FORCE** option is set in *options*. If the **BE\_DESTROY\_ORIGIN** option is set in *options*, the **be\_destroy()** function will destroy the origin snapshot to this boot environment as well.

The **be\_nicenum()** function will format *name* in a traditional ZFS humanized format, similar to **humanize\_number(3)**. This function effectively proxies **zfs\_nicenum()** from **libzfs**.

The **be\_mount()** function will mount the given boot environment. If *mountpoint* is **NULL**, a mount point will be generated in */tmp* using **mkdtemp(3)**. If *result* is not **NULL**, it should be large enough to accommodate **BE\_MAXPATHLEN** including the null terminator. the final mount point will be copied into it. Setting the **BE\_MNT\_FORCE** flag will pass **MNT\_FORCE** to the underlying **mount(2)** call.

The **be\_mounted\_at()** function will check if there is a boot environment mounted at the given *path*. If

*details* is not NULL, it will be populated with a list of the mounted dataset's properties. This list of properties matches the properties collected by **be\_get\_bootenv\_props()**.

The **be\_unmount()** function will unmount the given boot environment. Setting the BE\_MNT\_FORCE flag will pass MNT\_FORCE to the underlying mount(2) call.

The **libbe\_errno()** function returns the **libbe** errno.

The **libbe\_error\_description()** function returns a string description of the currently set **libbe** errno.

The **libbe\_print\_on\_error()** function will change whether or not **libbe** prints the description of any encountered error to stderr, based on *doprint*.

The **be\_root\_concat()** function will concatenate the boot environment root and the given boot environment name into *result*.

The **be\_validate\_name()** function will validate the given boot environment name for both length restrictions as well as valid character restrictions. This function does not set the internal library error state.

The **be\_validate\_snap()** function will validate the given snapshot name. The snapshot must have a valid name, exist, and have a mountpoint of /. This function does not set the internal library error state.

The **be\_exists()** function will check whether the given boot environment exists and has a mountpoint of /. This function does not set the internal library error state, but will return the appropriate error.

The **be\_export()** function will export the given boot environment to the file specified by *fd*. A snapshot will be created of the boot environment prior to export.

The **be\_import()** function will import the boot environment in the file specified by *fd*, and give it the name *be\_name*.

The **be\_prop\_list\_alloc()** function allocates a property list suitable for passing to **be\_get\_bootenv\_props()**, **be\_get\_dataset\_props()**, or **be\_get\_dataset\_snapshots()**. It should be freed later by *be\_prop\_list\_free*.

The **be\_get\_bootenv\_props()** function will populate *be\_list* with *nvpair\_t* of boot environment names paired with an *nvlst\_t* of their properties. The following properties are currently collected as appropriate:

| Returned name | Description                      |
|---------------|----------------------------------|
| dataset       | -                                |
| name          | Boot environment name            |
| mounted       | Current mount point              |
| mountpoint    | "mountpoint" property            |
| origin        | "origin" property                |
| creation      | "creation" property              |
| active        | Currently booted environment     |
| used          | Literal "used" property          |
| usedds        | Literal "usedds" property        |
| usedsnap      | Literal "usedrefreserv" property |
| referenced    | Literal "referenced" property    |
| nextboot      | Active on next boot              |

Only the "dataset", "name", "active", and "nextboot" returned values will always be present. All other properties may be omitted if not available.

The **be\_get\_dataset\_props()** function will get properties of the specified dataset. *props* is populated directly with a list of the properties as returned by **be\_get\_bootenv\_props()**.

The **be\_get\_dataset\_snapshots()** function will retrieve all snapshots of the given dataset. *snap\_list* will be populated with a list of *nvpair\_t* exactly as specified by **be\_get\_bootenv\_props()**.

The **be\_prop\_list\_free()** function will free the property list.

## DIAGNOSTICS

Upon error, one of the following values will be returned:

- BE\_ERR\_SUCCESS
- BE\_ERR\_INVALIDNAME
- BE\_ERR\_EXISTS
- BE\_ERR\_NOENT
- BE\_ERR\_PERMS
- BE\_ERR\_DESTROYACT
- BE\_ERR\_DESTROYMNT
- BE\_ERR\_BADPATH
- BE\_ERR\_PATHBUSY
- BE\_ERR\_PATHLEN
- BE\_ERR\_BADMOUNT
- BE\_ERR\_NOORIGIN
- BE\_ERR\_MOUNTED

- BE\_ERR\_NOMOUNT
- BE\_ERR\_ZFSOPEN
- BE\_ERR\_ZFSCLONE
- BE\_ERR\_IO
- BE\_ERR\_NOPOOL
- BE\_ERR\_NOMEM
- BE\_ERR\_UNKNOWN
- BE\_ERR\_INVORIGIN

**SEE ALSO**

bectl(8)

**HISTORY**

**libbe** and its corresponding command, `bectl(8)`, were written as a 2017 Google Summer of Code project with Allan Jude serving as a mentor. Later work was done by Kyle Evans <[kevans@FreeBSD.org](mailto:kevans@FreeBSD.org)>.