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 value of NULL may be used, indicating that **be_snaphot**() 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 *nvlist_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*>.