

BE library
for
Snap Upgrade

Design Specifications
Revision 0.8

Table of Contents

- 1.Introduction.....3
- 2.The Boot Environment.....3
 - 2.1BE dataset layout.....3
 - 2.2Booting a BE.....4
 - 2.3BE snapshots.....4
 - 2.4Cloning a BE.....5
 - 2.5Destroying a BE.....5
 - 2.6Renaming a BE.....6
 - 2.7Activating a BE.....6
 - 2.8Mounting/Unmounting a BE.....7
 - 2.9Initializing a BE.....7
- 3.BE and Snapshot Auto Cleanup.....7
- 4.Auto Naming Scheme.....8
 - 4.1Auto naming of BE snapshots.....8
 - 4.2Auto naming of BE clones.....8
 - 4.3Resetting the naming state for a renamed BE.....9
 - 4.4Caveats.....9
- 5.Installer Requirements.....10
 - 5.1Initial install a fresh system.....11
 - 5.2Installer BE Migration from UFS to ZFS.....12
- 6.Packaging Requirements.....13
 - 6.1“Live” packaging transactions.....13
 - 6.2“Deferred activation” packaging transactions.....13
- 7.BE CLI Requirements.....15
- 8.Zones requirements.....15
- 9.Implementation.....15
 - 9.1Library Interfaces.....16
- 10.Dependencies.....19
- 11.Issues.....19
- 12.References.....19
 - 12.1Document Revision History.....19

1. Introduction

Snap Upgrade provides the mechanism to safely update software by capturing system states and allowing previous system states to be re-enabled. It does this by creating snapshots and clones of the entire system image. Snap Upgrade is layered on top of ZFS and leverages its cloning and snapshot capabilities. The boot environment library provides a set of functions to access and manipulate these captured system states. This library is intended to be used by the installer (as part of initial install and upgrade), the Image Packaging system, and the `beadm(1M)` utility.

2. The Boot Environment

The definition of a boot environment (also called a BE) is an instance of a bootable OpenSolaris environment consisting of a root file system and, optionally, other file systems mounted underneath it (e.g. `/usr`, `/var`, `/export`). The root file system and all other file systems of the BE that contain system software are required to be zfs datasets.

2.1 BE dataset layout

All BE root datasets reside in a designated location in a root pool:

```
<pool>/ROOT
```

This location is reserved specifically for root datasets and should not be used for any other purpose. The datasets directly underneath this location are the BE root datasets themselves, and are identified with a `uuid` stored in a `zfs(1M)` user property called `"org.opensolaris.libbe:uuid"`. The names of those datasets correspond to the names of the BEs. For example:

```
<pool>/ROOT/myBE
```

```
<pool>/ROOT/otherBE
```

signifies there are two BEs, one named `"myBE"` and another named `"otherBE"`. All datasets underneath a root dataset are subordinate file systems for that root dataset and together with that root dataset, compose a boot environment. For example, a BE could have the following datasets:

```
<pool>/ROOT/myBE          mounted as /
```

```
<pool>/ROOT/myBE/opt      mounted as /opt
```

```
<pool>/ROOT/myBE/usr      mounted as /usr
```

```
<pool>/ROOT/myBE/var      mounted as /var
```

All datasets under the BE root dataset are considered private, and not shared with any other boot environment. When a boot environment is cloned, all datasets under the BE root dataset, including the root dataset itself, are cloned to create the new BE. The system administrator may create additional non-shared datasets for the boot environment under the

BE root dataset. For example, the system administrator may create the following datasets for the 'myBE' boot environment:

```
<pool>/ROOT/myBE/foo          mounted as /foo
<pool>/ROOT/myBE/usr/local    mounted as /usr/local
```

Datasets that reside outside the designated <pool>/ROOT area are shared between all BEs. They are seen and mounted identically regardless of which BE is currently booted. The only file systems located in the shared area should be user data file systems. The system administrator may create additional shared datasets for the boot environment in the shared area. For example:

```
<pool>/export                  mounted as /export
<pool>/export/home             mounted as /export/home
<pool>/shared                  mounted as /shared
```

2.2 Booting a BE

When a BE boots, its root dataset is mounted as the root file system, and all of the datasets hierarchically subordinate to its root dataset get processed by the `svc:/system/filesystem/minimal` SMF service to mount them in the correct location. The `svc:/system/filesystem/local` SMF service comes along after that to process the remaining datasets from the shared area (the datasets outside of the <pool>/ROOT area) by calling “zfs mount -a”. Since the zfs mountpoint property for all non-shared datasets of a BE are absolute paths, a non-shared dataset from some other inactive BE may have the same mountpoint as a non-shared dataset from the BE being booted. To prevent zfs from trying to mount the dataset from the inactive BE at the same location (which causes an error), the 'canmount' zfs property is set to 'noauto' so that “zfs mount -a” ignores it.

Thus, it is required that all non-shared BE datasets (any dataset in the <pool>/ROOT area) have their 'canmount' property set to 'noauto'.

2.3 BE snapshots

A BE snapshot is a coordinated set of snapshots of the datasets that compose a BE. The datasets included in a BE snapshot include only the non-shared datasets of the BE, not the shared datasets. A BE snapshot is taken using the recursive zfs(1M) snapshotting feature, hence the snapshot of each dataset is commonly named and ensured to be taken at the same moment in time. For example, the following BE named “myBE” has a BE snapshot called “today”

```
<pool>/ROOT/myBE              mounted as /
<pool>/ROOT/myBE@today
<pool>/ROOT/myBE/usr          mounted as /usr
<pool>/ROOT/myBE/usr@today
```

<pool>/ROOT/myBE/opt mounted as /opt
<pool>/ROOT/myBE/opt@today

2.4 Cloning a BE

A BE can be cloned from an existing BE or an existing BE snapshot. If cloned from an existing BE, a BE snapshot is taken of that existing BE at that point in time to create the clone from. Each non-shared dataset from the existing BE is cloned to create the new BE. For example, if there is an existing BE named “myBE” with the following non-shared datasets:

<pool>/ROOT/myBE
<pool>/ROOT/myBE/usr
<pool>/ROOT/myBE/usr/local

Cloning “myBE” to create “newBE” results in the following new datasets:

<pool>/ROOT/myBE@newBE
<pool>/ROOT/myBE/usr@newBE
<pool>/ROOT/myBE/usr/local@newBE
<pool>/ROOT/newBE
<pool>/ROOT/newBE/usr
<pool>/ROOT/newBE/usr/local

Upon cloning a BE, if any of the BE's non-shared datasets are set to be legacy mounted, the entries for those datasets in the new BE's /etc/vfstab file will be adjusted to reflect the new BE's name. An entry for the new BE will also be added to GRUB menu.

2.5 Destroying a BE

When a BE is destroyed, all of its non-shared datasets and optionally, any snapshots that those datasets may have, get destroyed. If any of the BE non-shared datasets have a dependent clone or clones, the BE is “demoted” before getting destroyed. The demotion process consists of finding the youngest snapshot that has a clone, and then promoting that clone. After this promotion, the dataset getting destroyed should have no dependent clones, and hence can be destroyed.

The origin snapshot parenting the BE datasets getting destroyed also get destroyed (as long as they don't have any other dependent clones.) No shared datasets (datasets outside of the <pool>/ROOT area) will get destroyed since they are in use by other BEs.

After destroying the datasets for the BE, the GRUB menu is also updated to remove the entry for the BE that was just destroyed. If the BE that was just destroyed happens to be the BE that was set to be the next active BE on reboot, the currently running BE is set to be the next active BE on reboot.

2.6 Renaming a BE

When a BE is renamed, the BE's root dataset is renamed to the new name. For example, if we have an existing BE named “myBE” with the following non-shared datasets:

```
<pool>/ROOT/myBE
<pool>/ROOT/myBE/usr
<pool>/ROOT/myBE/usr/local
```

Renaming this BE to “newBE” consists of simply renaming the root dataset to the new name:

```
<pool>/ROOT/newBE
<pool>/ROOT/newBE/usr
<pool>/ROOT/newBE/usr/local
```

A limitation of this is that the currently active BE cannot be renamed since its root dataset is currently mounted at “/”. Only inactive BEs can be renamed.

Upon renaming a BE, the BE is also checked to see if any of its non-shared datasets are set to be legacy mounted. If so, the entries for those datasets in the BE's `/etc/vfstab` file will be adjusted to reflect the new BE name. If the BE was the next active on reboot BE, the 'bootfs' property for the pool in which its root dataset resides will also get updated to reflect the new BE name. And finally, the BE's entry in the GRUB menu will have its 'bootfs' directive updated to reflect the new name.

2.7 Activating a BE

When a BE is activated, it will be set to be the next active BE upon reboot. This means that when the system is rebooted, by default and without user interaction, this BE will be what boots. The process of activating a BE includes:

- Promoting all of the BE's non-shared datasets such that they are at the top of the dependency chain.
- Ensuring the GRUB version installed on the pool in which this BE resides supports the capabilities of the software version installed in this BE. If it does not, install the GRUB loader from the BE being activated into the pool.
- Setting the 'bootfs' property in the pool in which this BE resides to this BE's root dataset.
- Updating the GRUB menu to make this BE's entry the default entry.

2.8 Mounting/Unmounting a BE

When an inactive BE is mounted at some altroot, all of the BE's non-shared datasets are mounted at their configured mountpoints, with respect to the altroot. For example, if the BE named “otherBE” has the following non-shared datasets, is mounted at `/a`, its datasets will get mounted at the following locations:

<pool>/ROOT/otherBE	mounted at /a
<pool>/ROOT/otherBE/usr	mounted at /a/usr
<pool>/ROOT/otherBE/usr/local	mounted at /a/usr/local
<pool>/ROOT/otherBE/var	mounted at /a/var

Optionally, all shared datasets can be made accessible (readwrite or readonly) to the mounted BE with respect to the alroot location. Since shared datasets are currently mounted for the live BE, they get loopback mounted onto their respective locations in the alroot.

2.9 *Initializing a BE*

When initializing a BE, an “empty” boot environment is created. That is, the datasets specified for the BE (both non-shared and shared) are created and left empty. An initialized BE can be mounted, unmounted, renamed, and destroyed but cannot be activated. Activation will fail for this BE because it will lack the set of appropriate files for it be considered a valid BE. BE initialization is a special procedure designed to be used by the initial installation process. Once the BE is installed into and has the appropriate files present, it can be activated. Note: the set of files required to be present for a BE to be considered valid are TBD.

3. BE and Snapshot Auto Cleanup

The design for boot environment and snapshot auto cleanup is TBD.

4. Auto Naming Scheme

When a BE or BE snapshot is created, the caller can pass in an explicit BE name or a snapshot name to create. This is the straightforward case, and the resultant BE or BE snapshot is created accordingly. However, the BE library also allows for the BE or the BE snapshot name to be left unspecified. When this occurs, the BE library uses an auto naming scheme to name the resultant BE or BE snapshot.

4.1 *Auto naming of BE snapshots*

An auto named BE snapshot will use the current date and time for the snapshot name. This helps the snapshot name stay unique, which prevents a case of running into snapshot naming conflicts when we promote a dependent BE clone of a BE. A BE snapshot name will be of the form:

<date>-<time>

For example, if we're starting out with a BE named “myBE” (root dataset lives at

"<pool>/ROOT/myBE"), which has one subordinate file system for /opt, then creating two snapshots of this BE with auto naming results in:

```
<pool>/ROOT/myBE
<pool>/ROOT/myBE@2008-02-13-10:28:36
<pool>/ROOT/myBE/@2008-02-13-10:29:04
<pool>/ROOT/myBE/opt
<pool>/ROOT/myBE/opt@2008-02-13-10:28:36
<pool>/ROOT/myBE/opt@2008-02-13-10:29:04
```

4.2 Auto naming of BE clones

The auto naming of BE clones is generated by appending an increment number to the base name of the original BE with a hyphen. An auto named BE snapshot is created for use as the origin of the auto named BE clone. For example, continuing on with the example above, creating an auto named BE clone of "myBE" results in:

```
<pool>/ROOT/myBE@2008-02-13-10:40:33
<pool>/ROOT/myBE-1
<pool>/ROOT/myBE/opt@2008-02-13-10:40:33
<pool>/ROOT/myBE-1/opt
```

Subsequent BE clones taken for "myBE" with auto naming recognizes the existing naming state, and simply increments the naming number. Continuing with this example, creating another clone of "myBE" at this point results in:

```
<pool>/ROOT/myBE@2008-02-13-10:45:10
<pool>/ROOT/myBE-2
<pool>/ROOT/myBE/opt@2008-02-13-10:45:10
<pool>/ROOT/myBE-2/opt
```

Subsequent BE clones taken for any BE that is part of the "myBE" name stream ("myBE-1" or "myBE-2") with auto naming also recognizes the existing naming state, and continues the increment naming. Continuing on from the above example, a clone of "myBE-1" would result in:

```
<pool>/ROOT/myBE-1
<pool>/ROOT/myBE-1@2008-02-13-10:46:35
<pool>/ROOT/myBE-3
<pool>/ROOT/myBE-1/opt
<pool>/ROOT/myBE-1/opt@2008-02-13-10:46:35
<pool>/ROOT/myBE-3/opt
```

4.3 *Resetting the naming state for a renamed BE*

The continuous increment number makes logical sense when we're dealing with a single BE name stream. However, when a BE is explicitly renamed to something else, it starts a new BE name stream so clones of that BE start a new auto naming state. Continuing with the above example, if the BE named "myBE-2" were renamed to "foo", then a BE clone of it results in:

```
<pool>/ROOT/foo
<pool>/ROOT/foo@2008-02-13-10:50:30
<pool>/ROOT/foo-1
<pool>/ROOT/foo/opt
<pool>/ROOT/foo/opt@2008-02-13-10:50:30
<pool>/ROOT/foo-1/opt
```

4.4 *Caveats*

A caveat to this naming scheme is that if a user explicitly creates a BE named "be-50", then cloning this BE with auto naming results in a BE named "be-51". This also means that if a user explicitly created a BE named with the same name as some existing auto named BE stream, then that user-created BE impacts the naming of the next auto named BE in that BE name stream. For example, continuing on with example above, if a user explicitly created a BE named "myBE-50", we have:

```
<pool>/ROOT/myBE
<pool>/ROOT/myBE@2008-02-13-10:28:36
<pool>/ROOT/myBE@2008-02-13-10:29:04
<pool>/ROOT/myBE@2008-02-13-10:45:10
<pool>/ROOT/myBE-1
<pool>/ROOT/myBE-1@2008-02-13-10:46:35
<pool>/ROOT/foo
<pool>/ROOT/myBE-3
<pool>/ROOT/myBE/opt
<pool>/ROOT/myBE/opt@2008-02-13-10:28:36
<pool>/ROOT/myBE/opt@2008-02-13-10:29:04
<pool>/ROOT/myBE/opt@2008-02-13-10:45:10
<pool>/ROOT/myBE-1/opt
<pool>/ROOT/myBE-1/opt@2008-02-13-10:46:35
<pool>/ROOT/foo/opt
<pool>/ROOT/myBE-3/opt
```

<pool>/ROOT/myBE-50

<pool>/ROOT/myBE-50/opt

If we now create an auto named BE clone of “myBE”, we end up creating “myBE-51”:

<pool>/ROOT/myBE@2008-02-13-11:55:23

<pool>/ROOT/myBE-51

<pool>/ROOT/myBE/opt@2008-02-13-11:55:23

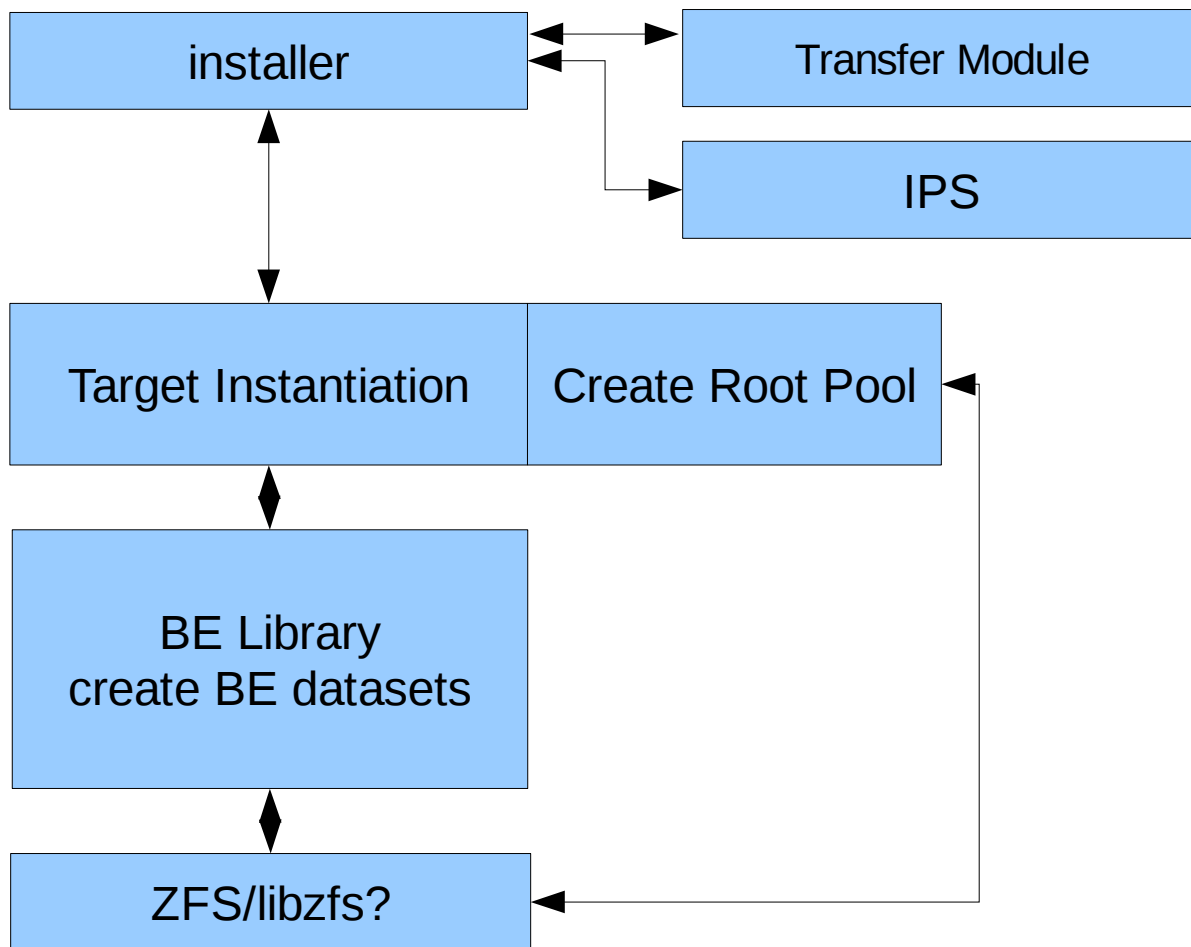
<pool>/ROOT/myBE-51/opt

The name, however, is purely aesthetic, and does not impact the BE library's ability to calculate BE dependencies.

5. Installer Requirements

The installer requirements fall into two areas - initial install and migration. For initial install, we need to support fresh install onto a zfs root pool. It is expected that the root pool will be created by the target instantiation module and passed in as an argument to libbe. The datasets needed for the BE will be created in that root pool. The migration case involves transitioning from a UFS based system to a zfs BE. In this case, archiving the existing installed system to an alternate location is required so that the existing disk can be reformatted to accommodate a zfs root pool and a zfs BE.

5.1 Initial install a fresh system



When installing a fresh system, an initial boot environment needs to be set up and made active. To do this several things need to happen. The first is to check for existing BEs on the system (the purpose of this check is to detect whether upgrade is possible, and to be able to warn the user what data is being destroyed on the disk if initial install is chosen.)

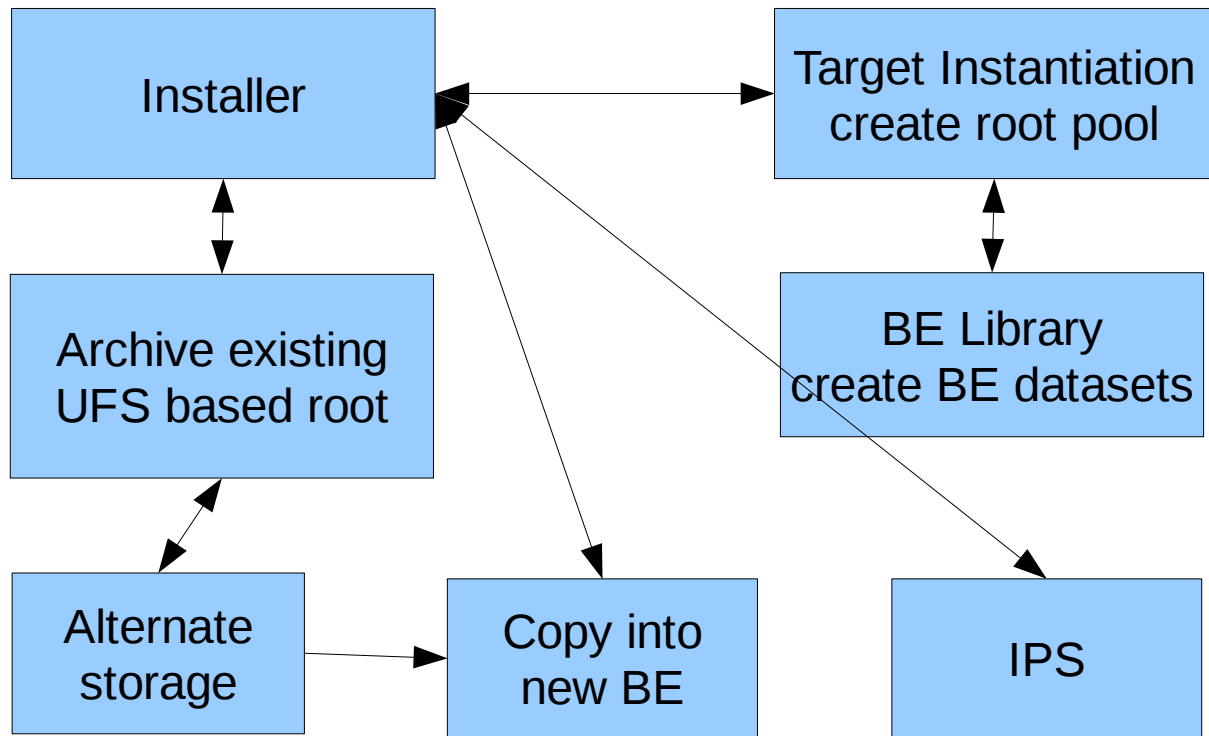
The next step is to layout a raw disk, possibly an fdisk table, and a vtoc label, and create a root pool and a zfs BE. The installer will be calling into target instantiation to do this.

The next is to determine the size of the BE being used for the install. The BE can then be created and passed to the transfer module to be installed into or upgraded if possible.

Requirements on libbe:

- Discovery of existing BE's
- Determine available size in the root pool
- Create a BE in the specified root pool (if it will fit)
- Determine size of an existing BE.
- Mount/unmount a BE on a specified mount point.

5.2 Installer BE Migration from UFS to ZFS



The installer will support the upgrade case where there is no available local disk space to create a zfs BE to migrate to. This case requires that the existing UFS based installed root be archived onto an alternate location so that the existing disk can be reformatted to accommodate creation of a zfs root pool and zfs BE. The installer must copy the existing installed image onto an alternate location. The installer will then call into target instantiation to reformat the existing disk and create the zfs root pool and an initial zfs BE (like it would for a fresh install). The installer will then copy the pre-upgraded data back onto the zfs BE, and the upgrade operation can be performed using IPS. Once the upgrade is complete the BE then needs to be activated (made the default booted BE) and the system rebooted.

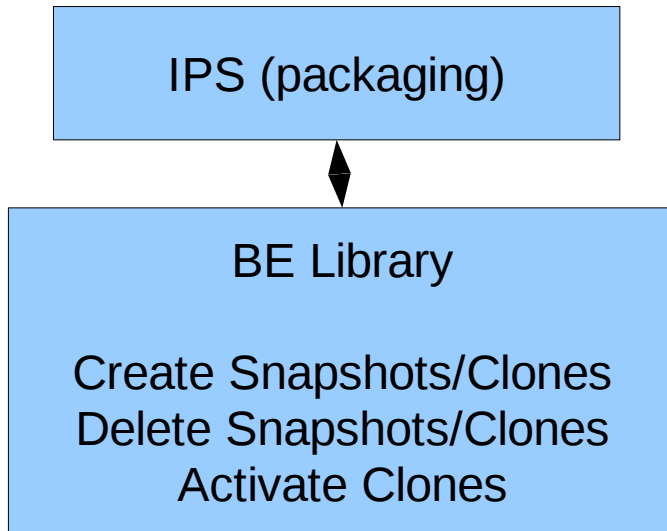
Requirements on libbe:

- Discovery of existing BE's
- Create a BE in a root pool.
- Ability to activate the BE so that it will be the BE booted on the next reboot.
- Mount/unmount a BE on a specified mount point.

6. Packaging Requirements

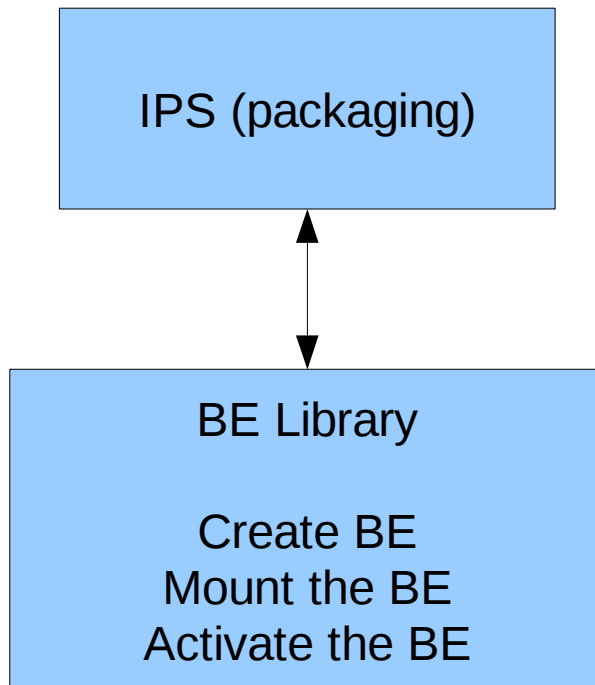
Package transactions can be put into two categories – live and deferred activation. For the “live” case, the transaction happens directly to the running system and for the “deferred activation” case, the transaction happens to a clone of the system, and a reboot is required for that packaging update to take effect.

6.1 “Live” packaging transactions



When applying a set of packages to a live system the packaging system needs to have a way to “rollback”, or return files to the state from before those packages were installed/updated. To do this the packaging system will call into the BE library to create a set of snapshots of the current BE's datasets. These snapshots can then be used as seen fit by the packaging system. For example the packaging system can use the original files contained in a snapshot to back-out a package by simply mounting the snapshot and copying the original files out of the snapshot on onto the current system.

6.2 “Deferred activation” packaging transactions

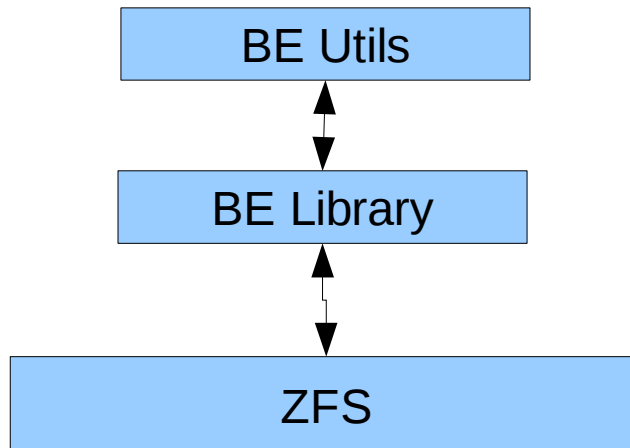


Package transactions that require deferred activation will be done to a clone of the system. The BE library will create a clone for the packaging utilities to operate on. This clone will essentially be a BE. Once the package transaction is complete, that BE will be set to be the next active BE upon reboot. As a consequence of being deferred, the rollback of this type of transaction will also require a reboot if its already been activated.

Requirements on libbe:

- Create a system clone (alternate BE)
- Provide the ability to mount/unmount a system clone.
- Activate an alternate BE.

7. BE CLI Requirements



The BE utilities will provide the command line interface to replace the functionality used in Live Upgrade. The BE utilities will call into the BE library to perform the following tasks.

- Create a BE
- Create a snapshot of a BE
- Delete a BE
- Delete a snapshot of a BE
- List BE details (and associated snapshots)
- List zone details (and associated snapshots)
- Mount/unmount a BE
- Activate a BE
- Rename a BE

8. Zones requirements

TBD.

9. Implementation

The BE library will be implemented as a shared library (called libbe). This is due to the fact we need to make use of other projects and libraries such as libzfs. For the Snap Upgrade project, the BE library will be implemented with ZFS support only.

9.1 Library Interfaces

The following function definitions are public interfaces. Functions that take an nvlist object

have their input and output attributes specified. The entire list of attribute definitions is specified in Table 1.

Function	be_errno_t be_list (char *be_name, be_node_list **)
Description	<ul style="list-style-type: none"> Provides BE configuration data for the BE named 'be_name'. If 'be_name' is NULL, probes all existing zfs pools and returns a list of all BE configurations in a be_node_list object.

Function	be_errno_t be_max_avail (char *root_pool, uint64_t *)
Description	<ul style="list-style-type: none"> Returns the max available size for a given root pool for creating a BE.

Function	be_errno_t be_init (nvlist_t *)
Description	<ul style="list-style-type: none"> Creates the datasets for a new BE and leaves them unpopulated. Resultant empty BE is mountable, but not bootable.
Input attributes	BE_ATTR_NEW_BE_NAME BE_ATTR_NEW_BE_POOL BE_ATTR_ZFS_PROPERTIES BE_ATTR_FS_NAMES BE_ATTR_FS_NUM BE_ATTR_SHARED_FS_NAMES BE_ATTR_SHARED_FS_NUM

Function	be_errno_t be_copy (nvlist_t *)
Description	<ul style="list-style-type: none"> Creates a new BE by cloning/copying some existing BE. If a snapshot name is also provided, it creates the new BE based on that snapshot of the existing BE. If origin BE name not specified, uses the currently running BE as the origin. If new BE name not provided, creates an auto named BE based on the origin BE name. If new BE's pool not provided, creates the new BE in the same pool as the origin BE. If the new BE is specified to be created in the same pool, zfs cloning is used; otherwise zfs_send/recv is used.
Input Attributes	BE_ATTR_ORIG_BE_NAME BE_ATTR_SNAP_NAME BE_ATTR_NEW_BE_NAME BE_ATTR_NEW_BE_POOL BE_ATTR_NEW_BE_DESC

	BE_ATTR_ZFS_PROPERTIES BE_ATTR_POLICY
Output Attributes	BE_ATTR_SNAP_NAME BE_ATTR_NEW_BE_NAME

Function	be_errno_t be_destroy (nvlist_t *)
Description	<ul style="list-style-type: none"> Destroys a BE, all of its subordinate datasets, and snapshots.
Input Attributes	BE_ATTR_ORIG_BE_NAME BE_ATTR_DESTROY_FLAGS

Function	be_errno_t be_activate (nvlist_t *)
Description	<ul style="list-style-type: none"> Makes the named BE the default booted BE on the next reboot of the system.
Input Attributes	BE_ATTR_ORIG_BE_NAME

Function	be_errno_t be_create_snapshot (nvlist_t *)
Description	<ul style="list-style-type: none"> Creates a snapshot of the named BE. If a specific snapshot name was passed in, will create snapshot with that name, otherwise, will create an auto named snapshot.
Input Attributes	BE_ATTR_ORIG_BE_NAME BE_ATTR_SNAP_NAME BE_ATTR_POLICY
Output Attributes	BE_ATTR_SNAP_NAME

Function	be_errno_t be_destroy_snapshot (nvlist_t *)
Description	<ul style="list-style-type: none"> Destroys the named snapshot of a named BE.
Input Attributes	BE_ATTR_ORIG_BE_NAME BE_ATTR_SNAP_NAME

Function	be_errno_t be_mount (nvlist_t *)
Description	<ul style="list-style-type: none"> Mounts a BE at a specified mountpoint Requires a valid mountpoint to be passed in
Input Attributes	BE_ATTR_ORIG_BE_NAME

BE_ATTR_MOUNTPOINT BE_ATTR_MOUNT_FLAGS

Function	be_errno_t be_unmount (nvlist_t *)
Description	<ul style="list-style-type: none"> • Unmounts a BE
Input Attributes	BE_ATTR_ORIG_BE_NAME BE_ATTR_UNMOUNT_FLAGS

Function	be_errno_t be_rename (nvlist_t *)
Description	<ul style="list-style-type: none"> • Renames a BE
Input Attributes	BE_ATTR_ORIG_BE_NAME BE_ATTR_NEW_BE_NAME

NAME	TYPE	DESCRIPTION
BE_ATTR_ORIG_BE_NAME	String	Name of origin BE
BE_ATTR_ORIG_BE_POOL	String	Name of origin BE pool
BE_ATTR_SNAP_NAME	String	Name of snapshot
BE_ATTR_NEW_BE_NAME	String	Name of new BE
BE_ATTR_NEW_BE_POOL	String	Name of new BE pool
BE_ATTR_POLICY	String	Name of BE policy type
BE_ATTR_ZFS_PROPERTIES	nvlist_t *	List of ZFS dataset properties
BE_ATTR_FS_NAMES	Array of strings	List of file system names to create under the BE root dataset
BE_ATTR_FS_NUM	uint16	Number of system file systems
BE_ATTR_SHARED_FS_NAMES	Array of strings	List of shared file systems to create
BE_ATTR_SHARED_FS_NUM	uint16	Number of shared file systems provided
BE_ATTR_MOUNTPOINT	String	Mountpoint for where to mount a BE
BE_ATTR_MOUNT_FLAGS	uint16	Flags for mounting a BE
BE_ATTR_UNMOUNT_FLAGS	uint16	Flags for unmounting a BE
BE_ATTR_DESTROY_FLAGS	Uint16	Flags for destroying a BE

Table 1: Attribute definitions for BE library

10. Dependencies

- Libzfs
 - Non-legacy root mount
 - canmount=noauto zfs property value
 - fs-* method script support

11. Issues

- Synclist – do we still support this in new BEs?
- zfs on-disk format upgrade – need detection, and a clever root pool migration scheme
- zfs version discovery – older version zfs can't read newer version zfs configurations
- Support for multiple root pools