

BE library  
for  
**Snap Upgrade**

Design Specifications  
Revision 0.7

# Table of Contents

1. Introduction.....	3
2. The Boot Environment.....	3
2.1 BE layout.....	3
2.2 BE snapshots.....	4
3. BE and BE Snapshot Auto Management.....	4
3.1 Auto Management Policy Attributes.....	4
3.2 Predefined Management Policy Types.....	5
3.2.1 The “volatile” policy type.....	5
3.2.2 The “static” policy type.....	5
4. Auto Naming Scheme.....	5
4.1 Auto naming of BE snapshots.....	6
4.2 Auto naming of BE clones.....	6
4.3 Resetting the naming state for a renamed BE.....	7
4.4 Caveats.....	7
5. Installer Requirements.....	9
5.1 Initial install a fresh system.....	9
5.2 Installer BE Migration from UFS to ZFS.....	10
6. Packaging Requirements.....	11
6.1 “Live” packaging transactions.....	11
6.2 “Deferred activation” packaging transactions.....	12
7. BE CLI Requirements.....	13
8. Zones requirements.....	13
8.1 Zones in BE root name space.....	13
8.2 Zones in a shared dataset.....	14
8.3 Zones containing private file systems.....	15
9. Library Interfaces.....	16
10. Dependencies.....	19
11. Issues.....	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. The Snap Upgrade system is layered on top of ZFS and leverages its cloning and snapshot capabilities. The BE 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, the BE utilities, and possibly other applications that require hooks into managing BEs.

## 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). The root file system and all other file systems of the BE which contain system software are required to be zfs datasets.

### 2.1 BE 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, therefore any random dataset placed underneath this location would be seen as a possible BE. 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 BE environment. For example, a BE could have the following datasets:

<pool>/ROOT/myBE	mounted as /
<pool>/ROOT/myBE/usr	mounted as /usr
<pool>/ROOT/myBE/var	mounted as /var
<pool>/ROOT/myBE/opt	mounted as /opt

Datasets that reside outside the designated <pool>/ROOT area are "shared" across all BEs. They are seen and mounted identically by a BE regardless of which BE is currently booted. These would be file systems that are not considered system file systems and would be things like /export. For example:

rpool/export	mounted as /export
rpool/export/home	mounted as /export/home

## 2.2 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 unshared 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
```

## 3. BE and BE Snapshot Auto Management

All BEs and BE snapshots will be created with a management policy type associated with them. For BEs, a zfs(1M) user property named 'com.sun.libbe:policy' will be used to store the management policy type. This property will be set in the root dataset for that BE. For BE snapshots, the policy type will be stored as part of the snapshot name itself. This management policy will dictate any default creation attributes and persistency of the BE snapshot/clone on the system. The caller will specify what type of management policy is associated with the BE or BE snapshot at creation time. There are two predefined management policy types defined. User defined types are not supported at this time.

The BE management policy will be actively enforced by a an smf(5) service. The configuration attributes of the service will be stored in the smf repository with the service definition and hence can be modified using the svccfg(1M) command.

### 3.1 Auto Management Policy Attributes

The set of attributes used to define an auto management policy are listed below with a description of how each attribute is used to determine when to automatically delete a BE or BE snapshot.

- **num\_min** – Specifies the minimum number of BEs or snapshots of this policy type that must exist before deleting the oldest. Set this value to '0' to ignore this attribute.
- **num\_max** – Specifies the maximum number of BEs or snapshots of this policy type to keep before deleting the oldest. **num\_min** must also be satisfied before deleting any BE or BE snapshot of this policy type. Set this value to '0' to ignore this attribute.
- **storage\_max** – Specifies the usage percentage level of the pool that a BE resides in before deleting the oldest in that pool. **num\_min** must also be satisfied before deleting any BE or BE snapshot of this policy type. Set this value to '0' to ignore this attribute.
- **delete\_snap** – When a BE is being deleted by the BE management service, this flag

specifies whether or not BE management can recursively delete all snapshots of the BE, as long as all of the snapshots are of the same policy type of the BE.

## 3.2 Predefined Management Policy Types

### 3.2.1 The “volatile” policy type

The “volatile” policy should be used for BEs and BE snapshots that are to be automatically cleaned up by the BE management service. The attribute values of the “volatile” policy type are stored as properties in the BE management service definition under the “volatile” property group. The system administrator can modify them to suit their needs. The default attribute values for the “volatile” policy are:

```
<property_group name='volatile' type='policy'>
  <!--default settings for the 'volatile' policy -->
  <propval
    name='num_max' type='count' value='24' />
  <propval
    name='num_min' type='count' value='6' />
  propval
    name='storage_max' type='count' value='70' />
  propval
    name='delete_snap' type='boolean' value='false' />
</property_group>
```

### 3.2.2 The “static” policy type

The “static” policy type is not defined in the BE management service definition, and hence is not tunable. This policy is equivalent to having all of the numeric policy attribute values set to '0'. BEs and BE snapshots under this policy would never get deleted automatically by the BE management service. During creation, if a policy type is not specified, all BEs and BE snapshots are created with this policy type. If a BE or BE snapshot lacks the policy definition, it gets treated with a “static” policy type. i.e. If a user manually used `zfs(1M)` to make a snapshot of any dataset of a BE, it would need to be explicitly destroyed.

## 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 falls back to using 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 have its policy type stored as part of the snapshot name. The current date and time will also be part of the snapshot name. This helps the snapshot name stay unique, which prevents a case of running into snapshot naming conflicts if we ever promote a dependent BE clone of a BE. A BE snapshot name will be of the form:

```
<policy>:<reserved>:<date>-<time>
```

The <reserved> component is currently not being used and is reserved for future use. It is currently left as the string, "-". 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@volatile:-:2008-02-13-10:28:36
<pool>/ROOT/myBE/@volatile:-:2008-02-13-10:29:04
<pool>/ROOT/myBE/opt
<pool>/ROOT/myBE/opt@volatile:-:2008-02-13-10:28:36
<pool>/ROOT/myBE/opt@volatile:-: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 to be the origin of the BE clone. For example, continuing on with the example above, creating a clone of "myBE" with auto naming results in:

```
<pool>/ROOT/myBE@volatile:-:2008-02-13-10:40:33
<pool>/ROOT/myBE-1
<pool>/ROOT/myBE/opt@volatile:-:2008-02-10-40-33
<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@volatile:-:2008-02-13-10:45:10
<pool>/ROOT/myBE-2
<pool>/ROOT/myBE/opt@volatile:-: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@volatile:-:2008-02-13-10:46:35
<pool>/ROOT/myBE-3
<pool>/ROOT/myBE-1/opt
<pool>/ROOT/myBE-1/opt@volatile:-: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@volatile:-:2008-02-13-10:50:30
<pool>/ROOT/foo-1
<pool>/ROOT/foo/opt
<pool>/ROOT/foo/opt@volatile:-: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 created 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@volatile:-:2008-02-13-10:28:36
<pool>/ROOT/myBE@volatile:-:2008-02-13-10:29:04
<pool>/ROOT/myBE@volatile:-:2008-02-13-10:45:10
<pool>/ROOT/myBE-1
<pool>/ROOT/myBE-1@volatile:-:2008-02-13-10:46:35
<pool>/ROOT/foo
<pool>/ROOT/myBE-3
<pool>/ROOT/myBE/opt
<pool>/ROOT/myBE/opt@volatile:-:2008-02-13-10:28:36
<pool>/ROOT/myBE/opt@volatile:-:2008-02-13-10:29:04
```

<pool>/ROOT/myBE/opt@volatile:-:2008-02-13-10:45:10  
<pool>/ROOT/myBE-1/opt  
<pool>/ROOT/myBE-1/opt@volatile:-: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”:

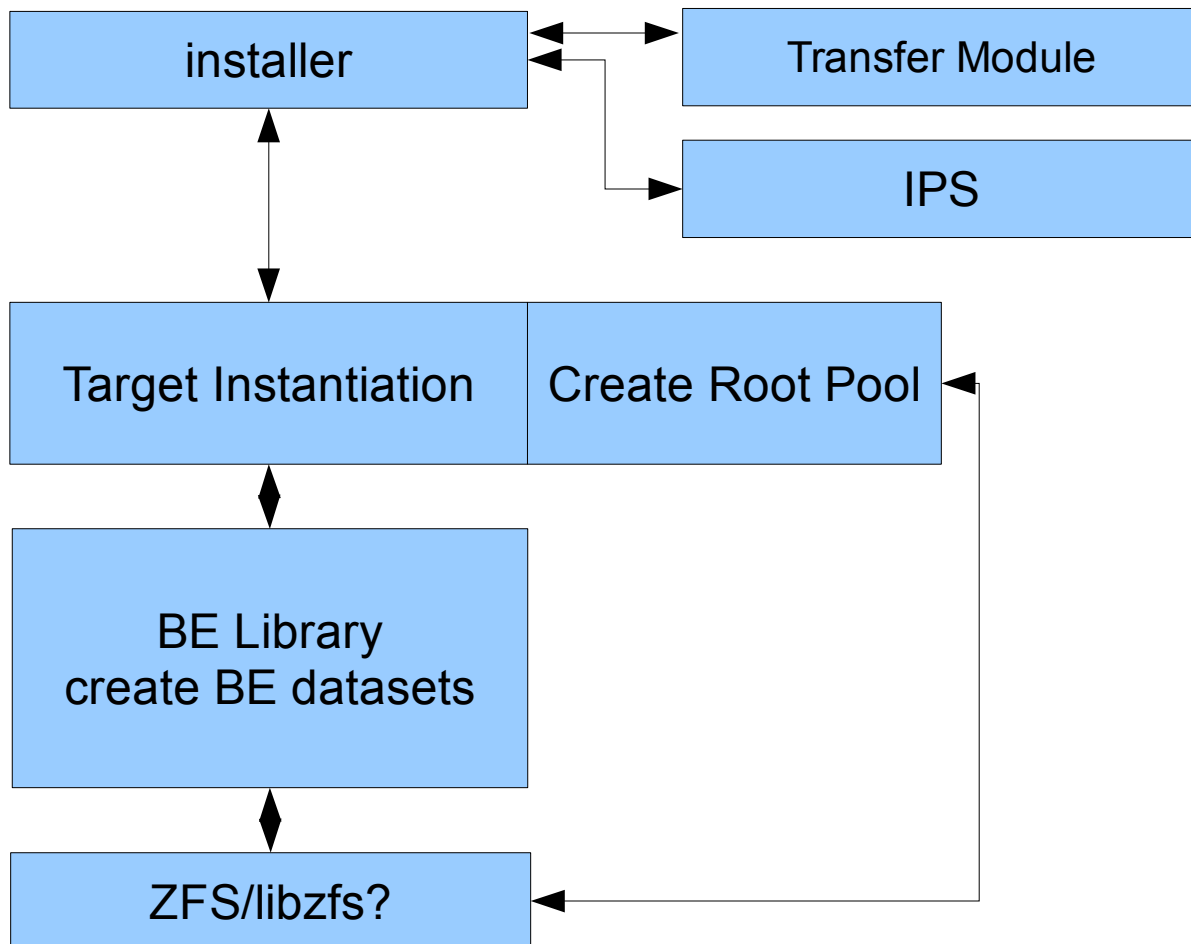
<pool>/ROOT/myBE@volatile:-:2008-02-13-11:55:23  
<pool>/ROOT/myBE-51  
<pool>/ROOT/myBE/opt@volatile:-:2008-02-13-11:55:23  
<pool>/ROOT/myBE-51/opt

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.

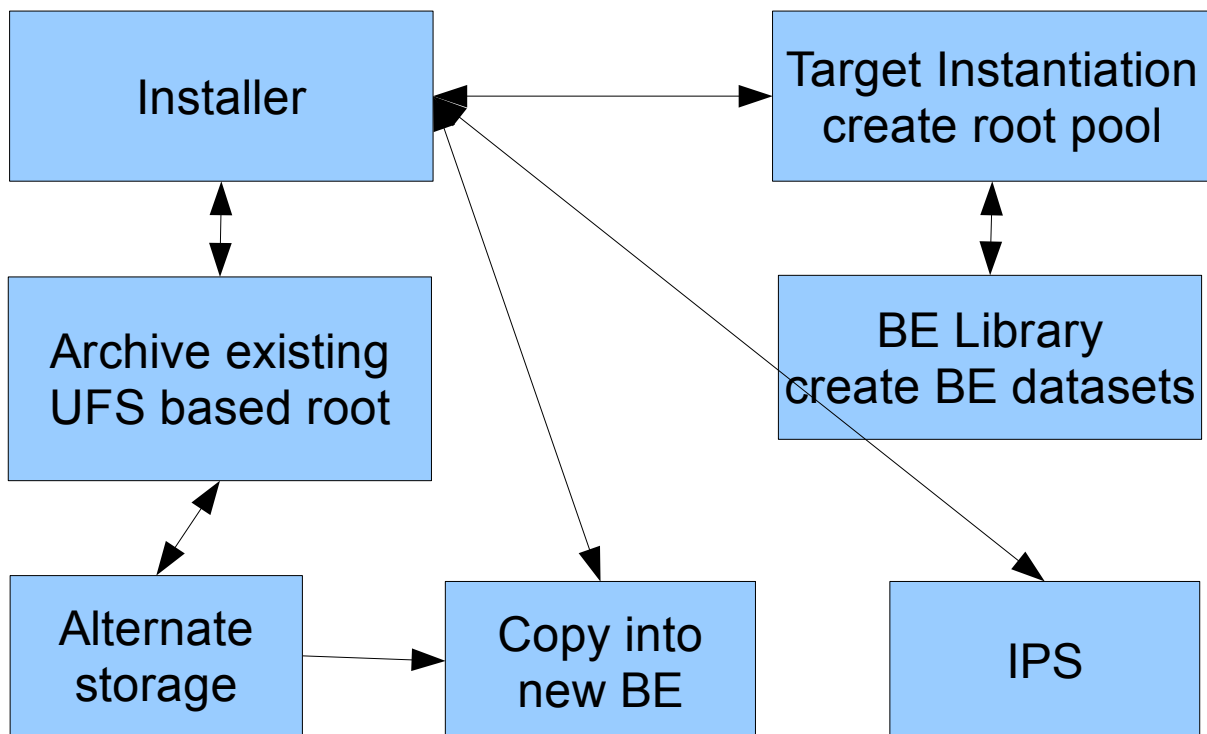
The next is to determine the size of the BE being used for the install. The BE can then be

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:

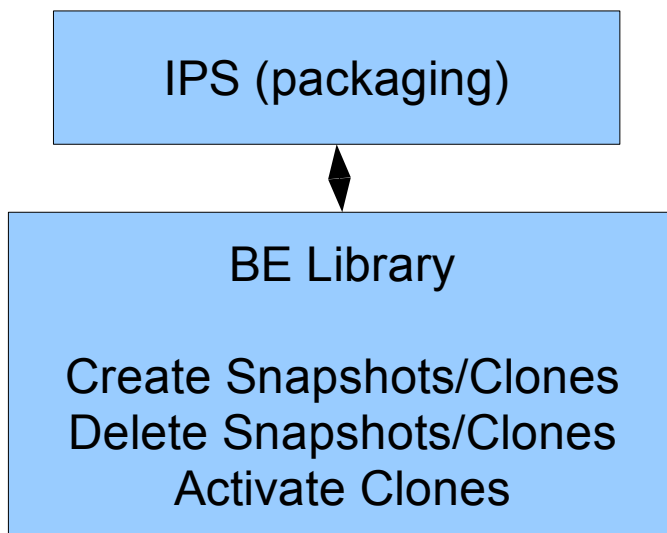
- Discovery of existing BE's
- Create a BE in a root pool.

- 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

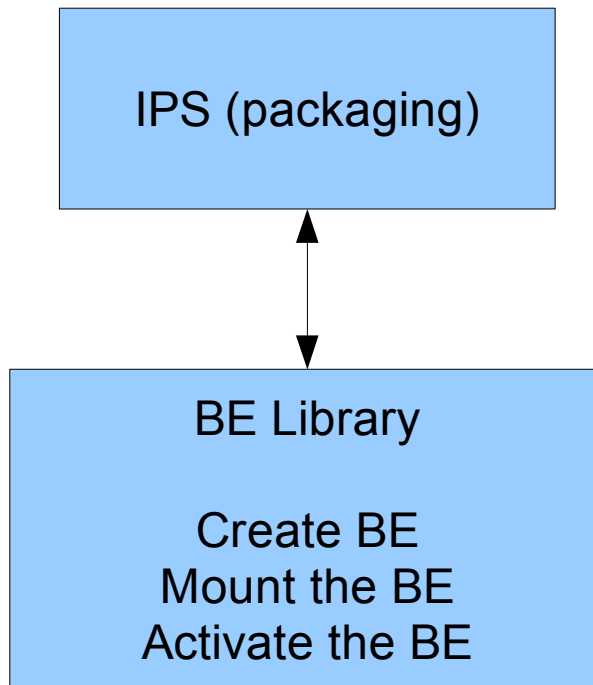
Package transactions can be put into two categories – live and deferred activation. For the “live” case, the transaction happens directly to the running system and for the “deferred activation” case, the transaction happens to a clone of the system, and a reboot is required for that packaging update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

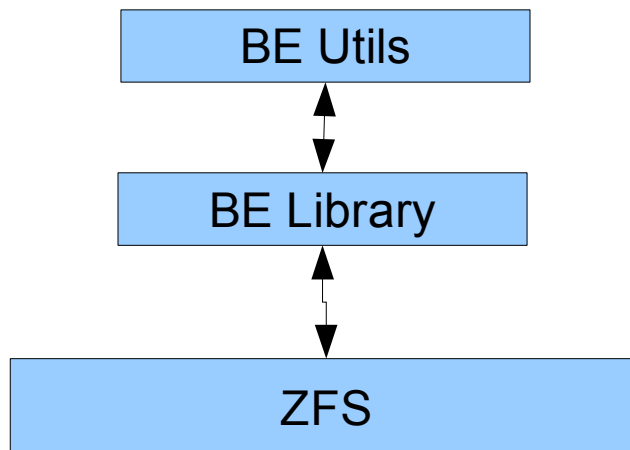


Package transactions that require deferred activation will be done to a clone of the system. The BE library will create a clone for the packaging utilities to operate on. This clone will essentially be a BE. Once the package transaction is complete, that BE will be set to be the next active BE upon reboot. 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

Should a BE contain zones, these zones will by default be created in the target BE depending on where they are located in the name space of the source BE. This section describes this interaction between where the zone is in the BE namespace and how it is copied into the new BE.

### 8.1 Zones in BE root name space

In this case the datasets will be cloned and the only change will be the path of the resultant datasets. The path will now contain the name of the target BE.

Existing source BE:

rpool/ROOT/BE1/

rpool/ROOT/BE1/usr  
rpool/ROOT/BE1/var  
rpool/ROOT/BE1/opt  
rpool/ROOT/BE1/zones/zone1

Target BE after creation:

rpool/ROOT/BE2/  
rpool/ROOT/BE2/usr  
rpool/ROOT/BE2/var  
rpool/ROOT/BE2/opt  
rpool/ROOT/BE2/zones/zone1

## **8.2 Zones in a shared dataset**

When a zone is configured in a shared file system it needs to be cloned and renamed. A SMF service will run before local file systems are mounted and will modify the mountpoints of the zone datasets so that the zone dataset corresponding to the BE being booted gets mounted at the right place.

Existing source BE:

rpool/ROOT/BE1/  
rpool/ROOT/BE1/usr  
rpool/ROOT/BE1/var  
rpool/ROOT/BE1/opt  
rpool/export  
rpool/export/zone1

Target BE after creation:

rpool/ROOT/BE2/  
rpool/ROOT/BE2/usr  
rpool/ROOT/BE2/var  
rpool/ROOT/BE2/opt  
rpool/export  
rpool/export/zone1 (mounted as /export/zone1-BE1)  
rpool/export/zone1-BE2 (mounted as /export/zone1)

rpool/export is a shared dataset which contains the Container zone1. zone1 must be cloned and will be named <zoneName-beName> as shown above. During boot, a SMF service will mount the cloned zone at /export/zone1. The shared zone that was cloned from BE1 will be renamed as described above.

### **8.3 Zones containing private file systems**

A zone can be configured with its own dedicated file system (via 'add fs' or 'add dataset' in zonecfg(1M)). When a zone contains a dedicated file system, copying the zone to a target BE needs to be handled via two different methods. The private file system needs to be setup as a private or shared dataset for the zone in the target BE. For example, a zone has a dedicated /var and /export/home file system. The /var file system shouldn't be shared, but the /export/home file system should be.

For the zones private file systems that are not shared, the dataset gets cloned and named something that is specific to the target BE. The zones configuration data in the target BE gets modified to denote which file system to use as the private file system. The following shows an example of the above scenario:

Existing source BE:

```
rpool/ROOT/BE1/  
rpool/ROOT/BE1/usr  
rpool/ROOT/BE1/var  
rpool/ROOT/BE1/opt  
rpool/export/zone1  
rpool/zone1_var      (mounted as /var for zone1)  
rpool/zone1_exporthome  (mounted as /export/home for zone1)
```

zone1 is a zone in a shared dataset. zone1\_var and zone1\_exporthome are private datasets for zone1.

Target BE after creation:

```
rpool/BE/BE2/  
rpool/ROOT/BE2/usr  
rpool/ROOT/BE2/var  
rpool/ROOT/BE2/opt  
rpool/export/zone1  
rpool/export/zone1-BE2  
rpool/zone1_var      (mounted as /var for zone1 in BE1)  
rpool/zone1_var-BE2  (mounted as /var for zone1 in BE2)  
rpool/zone1_exporthome  (mounted as /export/home for zone1 in both BE1 and BE2)
```

## 9. Library Interfaces

The BE library will be implemented as a shared library, called libbe. The following function definitions are the exported 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 <b>be_list</b> (char *be_name, be_node_list **)
Description	<ul style="list-style-type: none"> <li>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.</li> </ul>

Function	be_errno_t <b>be_free_list</b> (be_node_list *)
Description	<ul style="list-style-type: none"> <li>Frees the storage used by a be_node_list object</li> </ul>

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

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

Function	be_errno_t <b>be_copy</b> (nvlist_t *)
Description	<ul style="list-style-type: none"> <li>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.</li> <li>If origin BE name not specified, uses the currently running BE as the origin.</li> <li>If new BE name not provided, creates an auto named BE based on the origin BE name.</li> </ul>

	<ul style="list-style-type: none"> <li>● If new BE's pool not provided, creates the new BE in the same pool as the origin BE.</li> <li>● If the new BE is specified to be created in the same pool, zfs cloning is used; otherwise zfs_send/recv is used.</li> </ul>														
Input Attributes	<table> <tr><td>BE_ATTR_ORIG_BE_NAME</td><td>*optional</td></tr> <tr><td>BE_ATTR_SNAP_NAME</td><td>*optional</td></tr> <tr><td>BE_ATTR_NEW_BE_NAME</td><td>*optional</td></tr> <tr><td>BE_ATTR_NEW_BE_POOL</td><td>*optional</td></tr> <tr><td>BE_ATTR_NEW_BE_DESC</td><td>*optional</td></tr> <tr><td>BE_ATTR_ZFS_PROPERTIES</td><td>*optional</td></tr> <tr><td>BE_ATTR_POLICY</td><td>*optional</td></tr> </table>	BE_ATTR_ORIG_BE_NAME	*optional	BE_ATTR_SNAP_NAME	*optional	BE_ATTR_NEW_BE_NAME	*optional	BE_ATTR_NEW_BE_POOL	*optional	BE_ATTR_NEW_BE_DESC	*optional	BE_ATTR_ZFS_PROPERTIES	*optional	BE_ATTR_POLICY	*optional
BE_ATTR_ORIG_BE_NAME	*optional														
BE_ATTR_SNAP_NAME	*optional														
BE_ATTR_NEW_BE_NAME	*optional														
BE_ATTR_NEW_BE_POOL	*optional														
BE_ATTR_NEW_BE_DESC	*optional														
BE_ATTR_ZFS_PROPERTIES	*optional														
BE_ATTR_POLICY	*optional														
Output Attributes	<table> <tr><td>BE_ATTR_SNAP_NAME</td></tr> <tr><td>BE_ATTR_NEW_BE_NAME</td></tr> </table>	BE_ATTR_SNAP_NAME	BE_ATTR_NEW_BE_NAME												
BE_ATTR_SNAP_NAME															
BE_ATTR_NEW_BE_NAME															

Function	be_errno_t <b>be_destroy</b> (nvlist_t *)				
Description	<ul style="list-style-type: none"> <li>● Destroys a BE, all of its subordinate datasets.</li> <li>● If specified in the optional flag, all snapshots of the BE are also destroyed.</li> </ul>				
Input Attributes	<table> <tr><td>BE_ATTR_ORIG_BE_NAME</td><td>*required</td></tr> <tr><td>BE_ATTR_DESTROY_FLAGS</td><td>*optional</td></tr> </table>	BE_ATTR_ORIG_BE_NAME	*required	BE_ATTR_DESTROY_FLAGS	*optional
BE_ATTR_ORIG_BE_NAME	*required				
BE_ATTR_DESTROY_FLAGS	*optional				

Function	be_errno_t <b>be_rollback</b> (nvlist_t *)				
Description	<ul style="list-style-type: none"> <li>● Rolls back a BE and all of its children datasets to the named snapshot. If not BE name is specified, it assumes we're operating on the live system.</li> <li>● If the snapshot being rolled back has younger snapshots, those snapshots and their dependent clones will get destroyed in the process.</li> </ul>				
Input Attributes	<table> <tr><td>BE_ATTR_ORIG_BE_NAME</td><td>*optional</td></tr> <tr><td>BE_ATTR_SNAP_NAME</td><td>*required</td></tr> </table>	BE_ATTR_ORIG_BE_NAME	*optional	BE_ATTR_SNAP_NAME	*required
BE_ATTR_ORIG_BE_NAME	*optional				
BE_ATTR_SNAP_NAME	*required				

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

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

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

Function	be_errno_t <b>be_mount</b> (nvlist_t *)
Description	<ul style="list-style-type: none"> <li>Mounts a BE at a specified mountpoint</li> <li>Requires a valid mountpoint to be passed in</li> <li>If specified in the optional flag, loopback mounts all shared file systems for the BE</li> </ul>
Input Attributes	BE_ATTR_ORIG_BE_NAME *required BE_ATTR_MOUNTPOINT *required BE_ATTR_MOUNT_FLAGS *optional

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

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

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_NEW_BE_DESC	String	Description of new BE
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_DESTROY_FLAGS	uint16	Flags for destroying a BE

Table 1: Attribute definitions for BE library

## 10. Dependencies

- ZFS changes
  - Non-legacy root mount

## 11. Issues

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