Primitives

There are **four Hermes user primitives**: blobs, buckets, vBuckets, and traits.

Blobs are finite uninterpreted byte sequences. Blobs are treated as *atomic*, i.e., they can be manipulated only as a whole.

Buckets are *disjoint* collections of blobs.

vBuckets are "blob hubs," i.e., collections of links (spokes) to blobs across buckets.

Traits decorate vBuckets and can be used to trigger certain actions and transformations on blobs.

Contents

Blob

A blob is a finite sequence of *uninterpreted bytes*. Its only attribute is its length (number of bytes).

A blob is created through a PUT operation on a bucket during which the caller assigns it a unique (in bucket scope) name.

A *copy* of a blob's byte stream can be retrieved via a GET operation on the "hosting" bucket.

There is *no random access* to the bytes in a blob.

A blob cannot be associated with more than one bucket.

A blob can be linked to zero or more vBuckets.

Bucket

A **bucket** is a collection of *named* blobs. It has a unique (per Hermes instance) name and zero or more associated blobs.

Bucket handles are a *finite resource* and there is an upper limit to the number of open buckets.

Typical operations

`bkt` - bucket, `name` - unicode string, `data` - byte arrray, `ctx` - context

Operation	Synopsis	Returns	Comment
`(CREATE name ctx)`	Creates bucket name in `ctx`	Bucket handle or status	
`(OPEN name ctx)`	Opens bucket `name` in `ctx`	Bucket handle or status	
`(PUT bkt name data ctx)`	name}} or overwrites an existing blob on `bkt` from `data` in `ctx`.	Status	
`(GET bkt name ctx)`	Retrieves a copy of the content of blob `name` on `bkt` as a byte array (in memory) in `ctx`	Status	Caller provides buffer
`(GETV bkt pred ctx)`	"Vectorized" `GET` predicated by `pred`. `ctx` drives if single or multiple dest. buffers, or lazy version.	Status, iterator	Caller provides buffer
`(DELETE bkt name ctx)`	Deletes blob `name` from `bkt` in `ctx`, `ctx` controls error behavior.	Status	Maintain a linkage count on blobs. Defer dangling link removal to vBuckets.
`(TRANSFER bkt name bkt1 name1 ctx)`	Transfers blob `name` from `bkt` to `bkt1` as `name1` in `ctx`	Status	
`(RENAME bkt old-name new-name ctx)`	Renames blob `old-name` on `bkt` in `ctx`	Status	
`(GET-BLOB-NAMES bkt pred ctx)`	Returns a list of blob names filtered by `pred` in `ctx`.	List of strings or iterator	
`(GET-INFO bkt ctx)`	Returns bucket info at level-of-detail (LOD) in `ctx`	Status	
`(RENAME old-name new-name ctx)`	Renames bucket `old-name` in `ctx`	Status	
`(CLOSE bkt ctx)`	Closes `bkt` along with all its associated resources (?) in `ctx`.	Status	Invalidates handle
`(DESTROY bkt ctx)`	Destroys `bkt` in `ctx`, `ctx` controls "aggressiveness"	Status	

Questions

- What kind of metadata do we maintain for blobs?
- What if any kind of state/status does a bucket have?
- Can a bucket have traits?

vBucket

A **vBucket** is a collection of links to blobs. It can be decorated with zero or more traits.

The blobs linked to a vBucket can belong to different buckets.

To link (a second time) a previously linked blob has no effect.

Typical operations

`vbkt` - vBucket, `trt` - trait, `blob` - blob, `pred` - predicate, `ctx` - context

Operation	Synopsis	Returns	Comment
`(CREATE name ctx)`	Creates a vBucket `name` in `ctx`	vBucket handle or status	
`(LINK vbkt blob ctx)`	Links `blob` to `vbkt` in `ctx`	Status	
`(UNLINK vbkt blob ctx)`	Unlinks `blob` from `vbkt` in `ctx`	Status	
`(GET-LINKS vbkt pred ctx)`	Retrieves the subset of links satisfying `pred` in `ctx`	List of strings or iterator	
`(ATTACH vbkt trt ctx)`	Attaches `trt` to `vbkt` in `ctx`	Status	
`(DETACH vbkt trt ctx)`	Detaches `trt` from `vbkt` in `ctx`	Status	
`(GET-TRAITS vbkt pred ctx)`	Retrieves the subset of attached traits satisfying `pred` in `ctx`	List of strings or iterator	
`(DELETE vbkt ctx)`	Deletes a vBucket in `ctx`	Status	Decrements the links counts of blobs in buckets.

Questions

- What metadata do we maintain for links?
  - Creation order

Trait

An **attribute specification** consists of a name (Unicode string) and a type. A **trait schema** is a set of attribute specifications, where the attribute names are unique in the scope of the trait schema.

```
{
  <attribute spec1>,
  <attribute spec2>,
  ...,
  <attribute spec>
}
```

A **trait** is a set of key/value pairs conforming to a trait schema, i.e., the keys match the trait schema attribute names and the values are of the type prescribed by the attribute specification.

For brevity, we will always speak of 'attributes' and 'traits', and it should be clear from the context whether it is an attribute specification, an attribute, a trait or a trait schema.

Specific traits, such as vBucket traits, have *required* and *user-de*fined attributes , i.e.,

```
{
  <required attributes>,
  <user-defined attributes>
}
```

Typical operations

`trt` - trait, `value` - value, `name` - name, `ctx` - context

Operation	Synopsis	Returns	Comment
`(CREATE name ctx)`	Creates a trait `name` in `ctx`	Trait handle or status	
`(GET trt name ctx)`	Retrieves the the value of attribute `name` in `ctx`	Value	
`(SET trt name value ctx)`	Sets the value of attribute `name` of `trt` in `ctx`	Status	
`(DELETE trt ctx)`	Deletes a trait in `ctx`	Status	Deletion has no side-effects on vBuckets that have copies of this trait attached.

vBucket trait

A **vBucket trait** has required attributes for handling events raised by (de-)attaching a trait and (un-)link a blob:

```
{
  "OnAttach" : (ON-ATTACH-CALLBACK vbkt trt ctx),  // trait attach (to vBucket) event handler
  "OnDetach" : (ON-DETACH-CALLBACK vbkt trt ctx),  // trait detach (from vBucket) event handler
  "OnLink" :   (ON-LINK-CALLBACK vbkt blob ctx),   // blob link (to vBucket) event handler
  "OnUnlink" : (ON-UNLINK-CALLBACK vbkt blob ctx), // blob unlink (from vBucket) event handler
  <user-defined attributes>
}
```

Event handlers are executed in application processes.

The user-defined attributes contain trait specific metadata, e.g., compression level.

Linking the same blob twice has no effect. To re-execute the event handler one must first unlink and then re-link.

Questions

- What guarantees do we make regarding the event handling? Atomicity? Transaction?
- What potential conflicts are there and how do we resolve them?
