Multiversion encoding: encoding PDS4 structures into a filesystem

Filesystems have fairly strict structures. Each file or directory (except the root) has a single parent. Files and directories have one set of contents at a time. If you ignore the possibility of links, the elements of a filesystem form a tree.

PDS4 object are different. Different versions of a component (bundle, collection or product) can have different contents. Any version of a component can have multiple parents. Different versions may have different children. This makes it very difficult to encode multiple versions of PDS4 objects into a single archive filesystem.

Say you have a bundle `bob` with collection `cathy` and products `pete` and `polly`. Product `pete` version 1 has for a parent collection `cathy` version 1. Say now that `polly` is updated to version 2. Collection `cathy` and bundle `bob` must be updated to version 2 because `polly` has changed and `polly` is part of them. But `pete` remains at version 1.

So `pete` version 1 now has two different parents: `cathy` version 1 and `cathy` version 2. And `cathy` has different contents, depending on its version: `cathy` version 1 contain `polly` version 1 and `cathy` version 2 contains `polly` version 2.

It's the versioning of PDS4 structure/directory contents that is the difficult part. We get around that by putting the info on contents into a file. Once we do that, the rest can be handled by encoding filepaths.

Dollar-signs are illegal in PDS4 filenames, so we can use them freely in our encoding, knowing that there can be no nameclashes.

To convert a LIDVID to an archive directory path, extract the PDS4 object names from the LID (the bundle, collection and product names, some of which might be missing), use them as directory names, and add a final subdirectory of the form `v$`*VID*.

Using the above example, the path corresponding to `bob` version 2 (LIDVID `urn:nasa:pds4:bob::2.0`) would be `/bob/v$2.0` and the path corresponding to `pete` (LIDVID `urn:nasa:pds4:bob:cathy:pete::1.0`) would be `/bob/cathy/pete/v$1.0`. Any files (and possibly directories) belonging to that component live in that directory.

To convert an archive directory path to a LIDVID, look for a path segment of the form `v$`*n.n*. The portion of the path before that is the LID, that segment corresponds to the VID, and any part of the path after that corresponds to the filepath within the component.

Using the above example, the path `/bob/cathy/v$3.4/foo/bar.txt` corresponds to a file at the path `foo/bar.txt` within a PDS4 collection with LIDVID `urn:nasa:pds4:bob:cathy::3.4`.

Note: we don't currently use subdirectories within PDS4 components, but the standard allows them, so we're keeping the possibility open.

To find the contents of a PDS4 structure, look within the directory for its LIDVID. There you will find a file named `subdir$versions.txt`. Its contents will be a series of lines, each of which has a subcomponent name, whitespace, and a VID.

Using the above example, `cathy` version 1 contains `pete` and `polly`, both version 1. So in the directory `/bob/cathy/v$1.0`, you'll find a `subdir$versions.txt` file. `/bob/cathy/v$1.0/subdir$versions.txt` will contain the lines

```
pete 1.0
polly 1.0
```

On the other hand, `cathy` version 2 contains `pete` version 1 and `polly` version 2. So `/bob/cathy/v$2.0/subdir$versions.txt` will contain the lines

```
pete 1.0
polly 2.0
```

Note that contents in the PDS4 sense will *not* be contents in the filesystem sense.

Using the above example, the PDS4 product `pete` version 1 is part of PDS4 collection `cathy` version 2, but the contents of `pete` version 1 is in `/bob/cathy/pete/v$1.0` which is *not* within the directory `/bob/cathy/v$2.0` where `cathy` version 2 lives.

You can find the code that reads and writes `subdir$versions.txt` files in `pdart.fs.multiversioned.SubdirVersions.py`.