To be able to store multiple versions of files and multiple versions of *directories* into a single filesystem, we must use a scheme to encode the version numbers and directory contents into the filesystem. A collection label for the collection with the LIDVID `urn:nasa:pds:hst_09059:data_acs_flt::2.1` would be stored at `/hst_09059/data_acs_flt/v$2.1/collection_data.xml` and to find the contents of the collection, you have to look into a file stored at `/hst_09059/data_acs_flt/v$2.1/subdir$versions.txt`. This is neither obvious nor easily readable by humans and it's best to let software handle it.

I've written `pdart.fs.multiversioned.Multiversioned` to handle this. It acts as a wrapper around a `pyfilesystem2` virtual filesystem and lets you access the data in a few different ways. To use it, you create a virtual filesystem object containing the archive data (in the difficult-to-read format), and then use it to create a Multiversioned object.

```
fs = fs.osfs.OSFS(directory where the bundle archive lives)
mv = Multiversioned(fs)
```

There are two ways to view the data.

Single-version view

If you are interested in just a single version, you can look at the latest version of a LID, or at a specific LIDVID, past or present.

```
lid = pdart.pds4.LID(desired bundle LID)
vv = mv.create_version_view()
```

will give you a `VersionView`, a read-only virtual filesystem object that contains the latest version of the bundle's contents. Files within `vv` are found at the PDS logical path */bundle-id/collection-id/product-id/filepath*. To read the collection label for the `hst_09059/data_acs_flt` that appears in that bundle version, you could simply say

```
f = vv.open("/hst_09059/data_acs_flt/
collection_data.xml", "r")
```

although in the actual, archive filesystem, it is stored at `/hst_09059/data_acs_flt/v$2.1/collection_data.xml`. Similarly, to find the contents of the collection, you can ask:

```
contents = vv.listdir("/hst_09059/
data_acs_flt/")
```

and get the contents although in the actual, archive filesystem, the names of the contents of that version of the collection are stored in `/hst_09059/data_acs_flt/v$2.1/subdir$versions.txt`. In short, `VersionViews` let you focus on the version you're interested in and ignore the rest of the archive and the existence of versioning.

If you want to look at a particular version, not the current one, you can instead use a LIDVID and say

```
lidvid = pdart.pds4.LIDVID(desired bundle LIDVID)
vv = VersionView(mv, lidvid)
```

The dictionary-like view

`Multiversioned` also has a dictionary-like interface where the keys are LIDVIDs and the values are `pdart.fs.multiversioned.VersionContents` objects. Each `VersionContents` object works like a directory: while a directory contains files, their contents, and subdirectories, each `VersionContents` object contains files, their contents, and a list of sub-LIDVIDs that it contains.

So, given a `Multiversioned` object, if you want to look at the contents of the collection with a given LIDVID, you can say

```
lidvid = pdart.pds4.LIDVID(desired bundle
LIDVID)
contents = mv[lidvid]
sub_lidvids = contents.subcomponents
```

Rather than actually packing the files' bytes into the `VersionContents` object (they could take up gigabytes!), we pack a single `pyfilesystem2` virtual filesystem object into it with a list of filenames. (This gives us flexiblity as it could hold its data in memory if small, or on the disk if large.)

So to process the files that are contained in that LIDVID, you could say

```
fs = contents.fs
for filepath in contents.filepaths:
    with fs.open(filepath, "r") as f:
        do something with the file f
```

When you want to add a new bundle, collection or product version to an archive, you build a `VersionContents` object, a LIDVID for your new version, and say

```
contents =
VersionContents.create_from_lidvids(
    sub_lidvids, filesys, filepaths)
```

```
lidvid = pdart.pds4.LIDVID(desired LIDVID)
mv[lidvid] = contents
```

Each LIDVID can be assigned only once—that is, once a version is written, it never changed—but read from as many times as you like.

(`VersionContents` objects can be built to work with either LIDs or LIDVIDs. This can be ignored most of the time.)