Image storage

The uploaded image files are identified by a combination of URI and filename. The URI is used as the principal identifier so we don't need to worry about collisions if two people each upload an image named "image.jpg". The filename is retained so the user can use their browser to download their image from the system and it will be named as they expect it to be.

We wanted a way to store thousands of image files so they would not all be in the same directory. We took our inspiration from the PairTree folks, and modified their algorithm to suit our needs. The general idea is to store files in a multi-layer directory structure based on the URI assigned to the file.

Let's consider a file with this information:

URI	http://vivo.mydomain.edu/individual/n3156
Filename	`lily1.jpg`

In this example, we assume that VIVO's home directory is at `/usr/local/vivo`.

We want to turn the URI into the directory path, but the URI contains prohibited characters. Using a PairTree-like character substitution, we might store it at this path:

```
/usr/local/vivo/uploads/file_storage_root/http+==vivo.mydomain.edu=individual=n3156/lily1.jpg
```

Using that scheme would mean that each file sits in its own directory under the storage root. At a large institution, there might be hundreds of thousands of directories under that root.

By breaking this into PairTree-like groupings, we insure that all files don't go into the same directory. Limiting to 3-character names will insure a maximum of about 30,000 files per directory. In practice, the number will be considerably smaller. So then it would look like this:

```
/usr/local/vivo/uploads/file_storage_root/htt/p+=/=vi/vo./myd/oma/in./edu/=in/div/idu/al=/n31/56/lily1.jpg
```

But almost all of our URIs will start with the same namespace, so the namespace just adds unnecessary and unhelpful depth to the directory tree. We assign a single-character prefix to that namespace, using the `file_storage_namespaces.properties` file in the uploads directory, like this:

```
a = http://vivo.mydomain.edu/individual/
```

And our URI now looks like this:

```
a~n3156
```

Which translates to:

```
/usr/local/vivo/uploads/file_storage_root/a~n/315/6/lily1.jpg
```

So what we hope we have implemented is a system where:

- Files are stored by URI and filename.
- File paths are constructed to limit the maximum number of files in a directory.
- "Illegal" characters in URIs or filenames will not cause problems.
  - even if a character is legal on the client and illegal on the server.
- Frequently-used namespaces on the URIs can be collapsed to short prefix sequences.
- URIs with unrecognized namespaces will not cause problems.

By the way, almost all of this is implemented in `edu.cornell.mannlib.vitro.webapp.filestorage.backend.FileStorageHelper` and illustrated in `edu.cornell.mannlib.vitro.webapp.filestorage.backend.FileStorageHelperTest`