File Management

Table of Contents:

Overview

Dealing with a massive amount of files is not an easy task. Accessing the local file system might be sometimes slow, and can become a bottleneck when working with big data. For these reasons and many others, Catalog keeps track of the file structure and registers the folders and files in their database. Some of the advantages of doing so are the following:

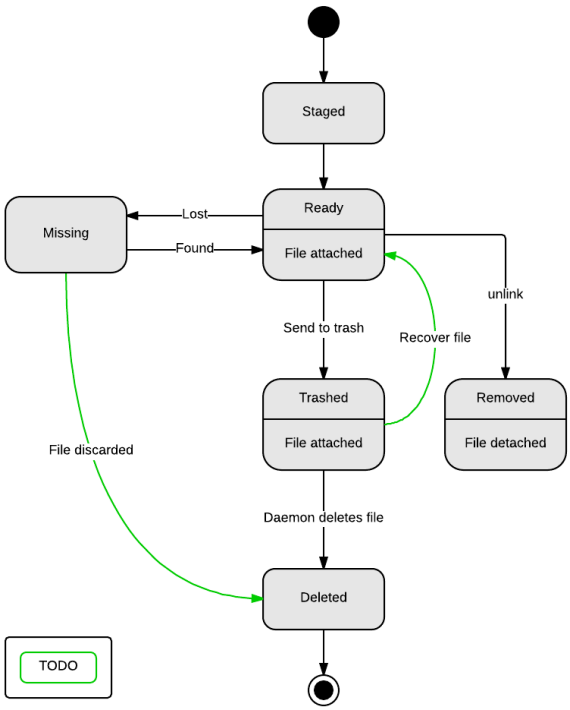
- Faster access to files.
- Users might add additional information, that is, adding custom annotations over the files.
- Catalog creates a different virtual file structure that does not have to be the same as in the file system. In fact, Catalog allows to group files or folders that can be placed in different paths within the file system into the same directory within the virtual Catalog file structure. This can be really convenient as explained in Link and Unlink section.
- Easier management of authorised users.

File status

Like it was mentioned before, Catalog will contain a file entry for each tracked file and folder from the study. Every file entry has a field "status" which represents the status of the entry. Valid status of a file entry are the following:

- **STAGE**: File entry has been created in Catalog, but the file is still not attached. This status only occurs when a user is uploading a file and should not last long.
- **READY**: General status of a file entry. Mean that everything is correct.
- **MISSING**: A file entry is set to missing when the file is no longer available or cannot be accessed from disk. If the file appears again, it will be set automatically to READY again.
- **TRASHED**: This status is similar to a file being in the rubbish bin. The file will no longer be returned although it will still be there. A file entry in this status can be safely restored again.
- **PENDING_DELETE**: This status is set when the user deletes a file. This status will remain until the file daemon :construction: starts the physical deletion. Next natural status would be DELETING.
- **DELETING**: This status is set when the running file daemon is actually doing the delete operation on disk.
- **DELETED**: This status is set when the file has been completely removed from disk.
- **REMOVED**: This status is only valid for external files. If the user wants to get rid of a linked file or diretory, he will have to run the unlink operation and this status will be set.

A basic life cycle diagram can be seen in the figure below:



Operations

Create

There are different ways of creating a file entry in Catalog:

- Upload file

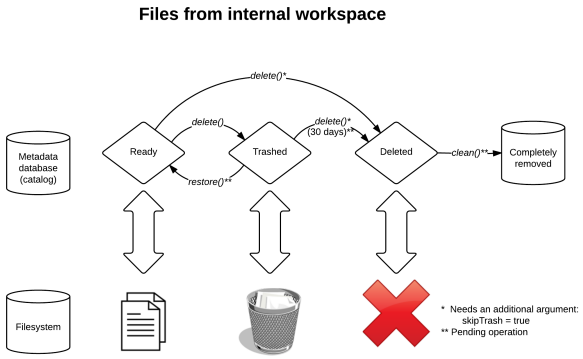
Uploading files is supported by Catalog, so a user can upload a local file to the server where OpenCGA is installed. The user will be able to choose the path where he want it to be placed. Every directory path will have associated a URI pointing to the physical location where the file will be located in the file system as explained in the link and unlink section.

- Link a file
- Create folder

The user is allowed to create as many directories as needed.

Delete and restore

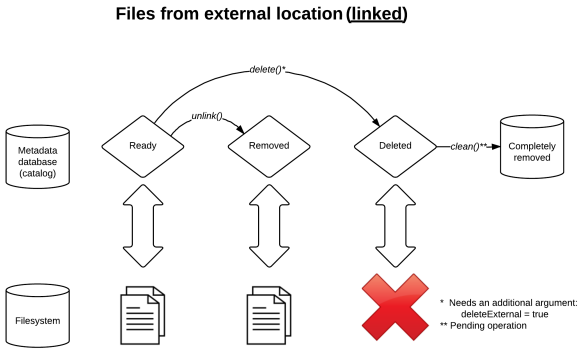
Deleting a file is one of the most critical operations in Catalog. For this reason, we have implemented the deletion into two different steps. By default, the delete operation will send the files to the rubbish bin. It will be possible to recover these files from the rubbish bin in the next release. However, this action do not touch the physical files in disk (if only, they are renamed to avoid possible collisions) so the files are still present in the system. A figure with the basic actions and status can be seen below, where we have the two different scenarios represented (the database and the file system).



A file can be deleted just by using the delete action. This action will send the file to the rubbish bin and set the status in Catalog to TRASHED. In the next release, the file daemon will completely delete the files from disk and set the status to DELETE 30 days after the files were sent to the bin. In any case, this will be the default behaviour and the admin will be able to modify the number of days or even disable this action. On the other hand, the user is able to force the deletion by adding the argument *skipTrash*.

Another daemon that is still pending to implement will be checking for the files that have been completely deleted from disk (DELETED) and will completely remove the old entries present in the database (clean action). Again, this will be easy to configure by the admin and even to disable.

Catalog also allows other files coming from more critical places to be available. These are the external files (explained in the next section). In this case, the deletion of files is even more critical and, normally, users would not expect their files to be deleted from the filesystem. For this reason, we don't allow deleting files that are not located within the Catalog boundaries unless the user is extremely sure of what he is doing adding an extra parameter as seen in the figure below.



In this case, users will be able to unlink a file or folder, which will be as if no file would have been linked (although we will still be keeping the file entry in the database) and the file/folder will still remain in their place. Only if the user is completely sure of what he is doing, he will be able to physically delete the file using the delete action with the extra parameter *deleteExternal*. If this parameter is not present and the user accidentally performed the delete operation, Catalog will just call the unlink method.

Warning

Restore operation is still pending. It will be one of the first operations to be implemented for the next release. Meanwhile, we encourage users to be extremely cautious when deleting a file.

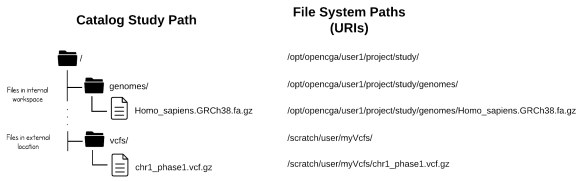
Link and Unlink

Each user has defined one single physical location where all the files and folders will be present by default. However, like it was mentioned in the overview, Catalog offers a new virtual file system by which users are allowed to place next to each other files and/or folders that are physically located in completely separated places within the file system.

Let's imagine that the admin decided that all the files for the user *user1* are going to be stored by default in /opt/opencga/user1. The *user1* in the example has one project called *project* and one study called *study* . Therefore, all the files that will be associated to that study of the user will be stored in /opt/opencga /user1/project/study by default.

However, that user has some vcf files in a concrete place of the cluster that he/she would like to use but don't want to move them. This action has been called **link**. By doing this, the user will be able to *link* the external folder or files anywhere into the Catalog virtual environment. After that, every time the user will query those files, it will be as if they were in the path /vcfs/*.

An example of this schema can be seen in the figure below.



Obviously, the user will be able to undo this action using the **unlink** action. This action will just remove the references to the files stored in Catalog so the /vcfs/ path could be used with something else but **it will not touch nor affect the physical files in any way**.

Share

Other important feature in Catalog is the ability to block or give access to concrete files to other registered users as well as giving or removing permissions to perform some actions with it. The complete list of file permissions can be seen here.

The mechanism to share and add new permissions is explained in the [[Sharing and Permissions page | Catalog Sharing]].