Basic Understanding of OCFL

This section introduces you to the basic structure and concepts behind the Oxford Common File Layout. This digital preservation specification was incorporated into Fedora 6.0 as a component of the persistence layer. You will learn more about the specification, anatomy, and benefits to implementing the OCFL in Fedora 6.x.

What is Oxford Common File Layout (OCFL)?

The OCFL is an application-independent specification that describes a simple, non-proprietary, specified, open-standards approach to the layout of preservation persistence. It provides a clear set of instructions for how files are laid out in the storage layer to better support digital preservation needs.

Why is Fedora 6.0 implementing OCFL?

One of the objectives of implementing OCFL in Fedora 6.0 was to enhance the digital preservation capabilities of the software. It will help to enhance long term digital preservation by generating a file structure and layout that mitigates the need for future content migrations (in the best case), and makes it easier to perform a content migration (in the worst case), thus another major benefit for users.

OCFL provides a simplified and transparent data structure that can help Fedora accomplish these goals. It is designed to promote long-term object management best practices within digital repositories.

Benefits of OCFL

- ### Parsability

  Because content in the OCFL format is laid out in a simple file and folder structure, it can be understood by both humans and machines without the original or proprietary hardware or software.

  Machine readability allows for simple applications, such as Fedora 6.0, to be placed on top of an existing OCFL storage root. If an application fails, it is possible to recover all the content, as long as you have the file system.

  Here is an example of the Flat Direct Storage layout:

  **Note, for performance reasons, a hierarchical/pairtree storage layout is recommended for production-scale OCFL storage roots. See: OCFL Extensions for community examples:

  ```
  [Storage_root]
      0=ocfl_1.0  (Declare that the root confirms to a version of OCFL)
      ocfl_1.0.txt  (Human-readable text of the OCFL specification; optional)
      ocfl_layout.json  Description of storage hierarchy layout; optional)
      bj102hs9687  (Object root)
               0=ocfl_object_1.0  (Declares that the object is an OCFL file)
                   inventory.json  (Inventory file)
                   inventory.json.sha512  (Inventory digest)
                   v1/  (Version directories; if this file has more versions, there would be v2, v3, and
  so on.)
                            inventory.json (Inventory file; if this is the most current version, it would
  be identical to the one at the object level)
                            inventory.json.sha512 (Inventory digest)
                            content/ (Payload)
                                    image.tiff
  ```

  This video provides a detailed explanation on the anatomy of OCFL.

  Watch from 30:00 - 37:20 (7 mins 20 secs) to learn more:

- ### Robustness

  Strong fixity is built into OCFL, which means it is able to guard against corruption. The inventory.json files lay out all the content and checksums, so it can be used to verify the integrity of the content of an object, and the object can be completely self contained.

  Here is an example of an inventory.json file:

```
{
  "id": "o3",
  "type": "https://ocfl.io/1.0/spec/#inventory",
  "digestAlgorithm": "sha512",
  "head": "v2",
  "contentDirectory": "content",
  "fixity": {},
  "manifest": {

"70ffe50550ae07cd0fc154cc1cd3a47b71499b5f67921b52219750441791981fb36476cd478440601bc26da16b28c8a2be4478b36091f26
15ac94a575581902c": [
      "v2/content/dir2/file3"
    ],

"9c614ba0d58c976d0b39f8f5536eb8af89fae745cbe3783ac2ca3e3055bb0b1e3687417a1d1104288d2883a4368d3dacb9931460c6e5231
17ff3eaa28810481a": [
      "v1/content/file1"
    ]
  },
  "versions": {
    "v1": {
      "created": "2019-08-05T15:57:53Z",
      "message": "commit message",
      "user": {
        "name": "Peter",
        "address": "peter@example.com"
      },
      "state": {

"9c614ba0d58c976d0b39f8f5536eb8af89fae745cbe3783ac2ca3e3055bb0b1e3687417a1d1104288d2883a4368d3dacb9931460c6e5231
17ff3eaa28810481a": [
          "file1"
        ]
      }
    },
    "v2": {
      "created": "2019-08-05T15:57:53Z",
      "message": "2",
      "user": {
        "name": "Peter",
        "address": "peter@example.com"
      },
      "state": {

"70ffe50550ae07cd0fc154cc1cd3a47b71499b5f67921b52219750441791981fb36476cd478440601bc26da16b28c8a2be4478b36091f26
15ac94a575581902c": [
          "dir2/file3"
        ],

"9c614ba0d58c976d0b39f8f5536eb8af89fae745cbe3783ac2ca3e3055bb0b1e3687417a1d1104288d2883a4368d3dacb9931460c6e5231
17ff3eaa28810481a": [
          "file1"
        ]
      }
    }
  }
}
```


- **Versioning**

OCFL was built with versioning capabilities - changes to objects can be recorded as new versions, allowing their history to persist and be recreated if needed. Changes to the objects are tracked over time and OCFL uses “forward delta” version control to reduce the amount of content being stored. A new version directory is created whenever a new version of an object is created, while files that have not changed stay in the original version folder. This helps save space and ensures files are not duplicated.

OCFL also subscribes to the concept of immutable versions, which aligns with preservation best practices. This means that a version, once created, should not be changed. By maintaining a version’s immutability it allows each version to exist as a snapshot in time.

To retrieve a version of an object, it can be reconstructed using the inventory.json file, which acts as instructions to recreate a version of an object.

See above code sample for an example of the versioning structure beginning at the "versions" line.

- **Storage diversity**

  The OCFL specification does not dictate the storage media you can use. It is designed to work with various storage infrastructures, including conventional file systems, object storage, and cloud-based offerings, such as Amazon S3.

- **Completeness**

  Within an OCFL repository the complete intellectual object is stored together with its metadata, so you can rebuild your entire repository simply from the files on disk.

  OCFL falls in line with Trusted Digital Repositories (TDR, ISO 16363), NDSA Levels of Preservation, and Open Archival Information Systems (OAIS). While these standards provide the recommended best practices for repositories and their administration, OCFL provides a clear set of instructions on how to layout files and folders on disk to better support digital preservation.

How is data written as OCFL objects in Fedora 6.0?

This demonstration shows how Containers are created using the REST API, and the corresponding data is written to disk as Oxford Common File Layout (OCFL) Objects.

Watch from 0:27 - 8:03 (7 mins 56 secs) to learn more:

For more information about Fedora OCFL Object Structure, visit the Fedora 6.0 wiki page.

Additional Resources

To learn more about OCFL, visit https://ocfl.io/, where you can find the most current specification, recent version release notes, tools, and how to access community developed implementations.

OCFL is developed and maintained by an editorial board, and not specific to Fedora. Fedora 6.x uses a java-implementation of the OCFL specification and as such is considered part of the OCFL-java community implementation group. Here is a list of other systems and institutions who are implementing OCFL: https://github.com/OCFL/spec/wiki/Implementations

Related articles

- Basic Understanding of OCFL
- Islandora - Test Migration
- Custom Fedora 3.x - Test Migrations
- Gather Functional Requirements
- Islandora - Install Migration Tools