Tutorial 1: Introduction to Fedora

Tutorial 1: Introduction to Fedora

Author: The Fedora Development Team

Purpose: This tutorial introduces the basic development questions, design concepts and project goals of the Flexible Extensible Digital Object Repository Architecture (Fedora).

Audience: This tutorial is intended for anyone who will be using the Fedora software in any capacity, or who is generally interested in Fedora and its development.

Table of Contents

Table of Figures

Section 1: What is Fedora?

What is Fedora?

Fedora is an acronym for Flexible Extensible Digital Object Repository Architecture. Fedora's flexibility makes it capable of serving as a digital repository for a variety of use cases. Among these are digital asset management, institutional repositories, digital archives, content management systems, scholarly publishing enterprises, and digital libraries. Fedora is open-source software licensed under the Mozilla Public License.

Fedora History

Fedora began in 1997 as a DARPA and NSF funded research project at Cornell University, where the initial reference implementation was developed by Sandra Payette, Carl Lagoze, and [first name] Dushay. Work at Cornell included a CORBA-based technical implementation, work on policy enforcement, and extensive interoperability testing with CNRI.

The first practical application of Fedora was the digital library prototype developed at UVa by Thornton Staples and Ross Wayland in 1999, where the software was adapted to the web and an RDBMS was added to improve performance. The initial work done on the Fedora prototype included scalability testing for 10 million objects.

A full scale development project was begun in 2002 with grant funding from the Andrew W. Mellon Foundation. This project's charge was to create a production quality Fedora system, using XML and web services to deliver digital content. Fedora 1.0 was released in May 2003, with subsequent releases following approximately every quarter which have added functionality and corrected bugs discovered by users and the Fedora development team.

In June 2004, the Andrew W. Mellon Foundation funded Fedora Phase 2 for an additional 3 year project. Planning for Phase 2 development is underway as of this writing.

Section 2: Motivation

This section provides information on the issues and questions that have motivated the development of Fedora.

The Problem of Digital Content

Digital content is not just documents, nor is it made up exclusively of the content from digital versions of currently owned non-digital content.

> **Conventional Objects:** books and other text objects, geospatial data, images, maps
>
> **Complex, Compound, Dynamic Objects:** video, numeric data sets and their associated code books, timed audio

As users become more sophisticated at creating and using complex digital content, digital repositories must also become more sophisticated. As digital collections grow, and are made use of in previously unconsidered ways, repository managers are faced with management tasks of increasing complexity. Collections are being built which contain multiple data types, and organizations have discovered a need to archive and preserve complex objects like those listed above, as well as web sites and other complex, multi-part documents. And finally, as collections grow in both size and complexity, the need to establish relationships between data objects in a repository becomes more and more apparent.

Key Research Questions

After considering the complexities of digital content, the original developers of Fedora at Cornell University developed five key research questions.

- How can clients interact with heterogeneous collections of complex objects in a simple and interoperable manner?
- How can complex objects be designed to be both generic and genre-specific at the same time?
- How can we associate services and tools with objects to provide different presentations or transformations of the object content?
- How can we associate specialized, fine-grained access control policies with specific objects, or with groups of objects?
- How can we facilitate the long-term management and preservation of objects?

Fedora Goals

These five key research questions led logically to ten goals for Fedora's development.

- **Identifiers:** provision of persistent identifiers; unique names for all resources without respect for machine address
- **Relationships:** support for relationships between objects
- **Tame Content:** normalization of heterogeneous content and metadata based on an extensible object model
- **Integrated Management:** efficient management by repository administrators not only of the data and metadata in a repository, but also of the supporting programs, services and tools that make presentation of that data and metadata possible
- **Interoperable Access:** provision of interoperable access by means of a standard protocol to information about objects and for access to object content; discovery and execution of extensible service operations for digital objects
- **Scalability:** provision of support for >10 million objects
- **Security:** provision of flexible authentication and policy enforcement
- **Preservation:** provision to features to support longevity and archiving, including XML serialization of objects and content versioning
- **Content Recon:** reuse of objects including object content being present in any number of contexts within a repository; repurposing of objects allowing dynamic content transformations to fit new presentations requirements
- **Self-Actualizing Objects:** ability of digital objects to disseminate launch-pads or tools for end-user interaction with content

Design Advantages—Where the Rubber Hits the Road

Fedora's design offers many advantages to repository developers and administrators.

Fedora's Digital Object Model

The digital object model offers the strengths and advantages of

- **Abstraction:** The object model is the same whether the object is data, behavior definitions, or behavior mechanism. It also does not matter what kind of data the digital objects is representing—text, images, maps, audio, video, geospatial data are all the same to Fedora.
- **Flexibility:** Implementers of Fedora can design their content models to best represent their data and the presentation requirements of their specific use case.
- **Generic:** Metadata and content are tightly linked within the digital object.

- **Aggregation:** Fedora objects can refer to data that is stored locally or that is stored on any web accessible server.
- **Extensibility:** Fedora's behavior interfaces are extensible because services are directly associated with data within a Fedora object. As the services change, the objects change along with them.

Distributed Repositories

The Fedora Architecture, as originally designed by Payette and Lagoze, was intended to support distributed repositories. This vision is described in the Fedora Specification and placeholders exist in the current software to build this functionality. Repository federation is important for several reasons. First, federation is a natural requirement for delivering integrated access to digital resources that are owned or managed by several institutions. Second, federation makes it easy for digital library and other applications to interface with multiple information sources in a seamless manner. Third, federation can help with scalability or performance issues for very large repositories. Specifically, a local federation of repositories can be established as a means of distributing load and object storage among several running repository instances, so that together these separate instances can be treated as one 'virtual repository.'

Preservation & Archiving

Fedora's archival and preservation capabilities include

- **XML:** Fedora objects' XML and the schema upon which they are based are preserved at ingest, during storage, and at export.
- **Content Versioning:** Fedora repositories offer implementers the option of versioning data objects. When a data object is versioned, the object's audit trail is updated to reflect the changes made to the object, when the change was made and by whom and a new version of the modified data is added to the object's XML. This new datastream cascades from the original and is numbered to show the relationship between original and version. This allows users to retrieve older versions of a data object by performing a date/time search and retrieval, or the most current version if the date/time criteria are not included in the search.
- **Object to Object Relationships:** Relationships between objects can be stored via the metadata included in the objects. This allows implementers to link together related objects into parent/child relationships.
- **Event History:** Every object in a Fedora repository contains an audit trail, which preserves a record of every change made to the object.

Content Repurposing

Application of different stylesheets to the data and metadata of a Fedora object allows multiple views of the object's content and metadata; for example, one view for a domain scholar and another for a K-12 audience. Additionally, content referenced in a Fedora data object can be dynamically transformed as it is called by a user by use of custom disseminators. Because of this inherent strength and flexibility, new views and data transformations are simple to add over time as the implementer's and user's requirements change.

Web Services

Fedora is exposed via web services and can interact with other web services. The interfaces and XML transmission are defined in WSDL.

Easy Integration with Other Applications and Systems

Fedora is not architected in such a way that any particular workflow or end-user application is assumed. Thus Fedora is able to function as a generic repository substrate upon which many kinds of applications can be created and to take advantage, over time, of advances in web services. This ability to grow and change builds longevity into a digital library system built on the Fedora architecture.

Section 3: Digital Object Model

Data Object

A data object in a Fedora repository describes content (data and metadata) and a set of associated behaviors or services that can be applied to that content. Data objects comprise the bulk of a repository.

The diagram below shows the architectural view of a Fedora digital object.

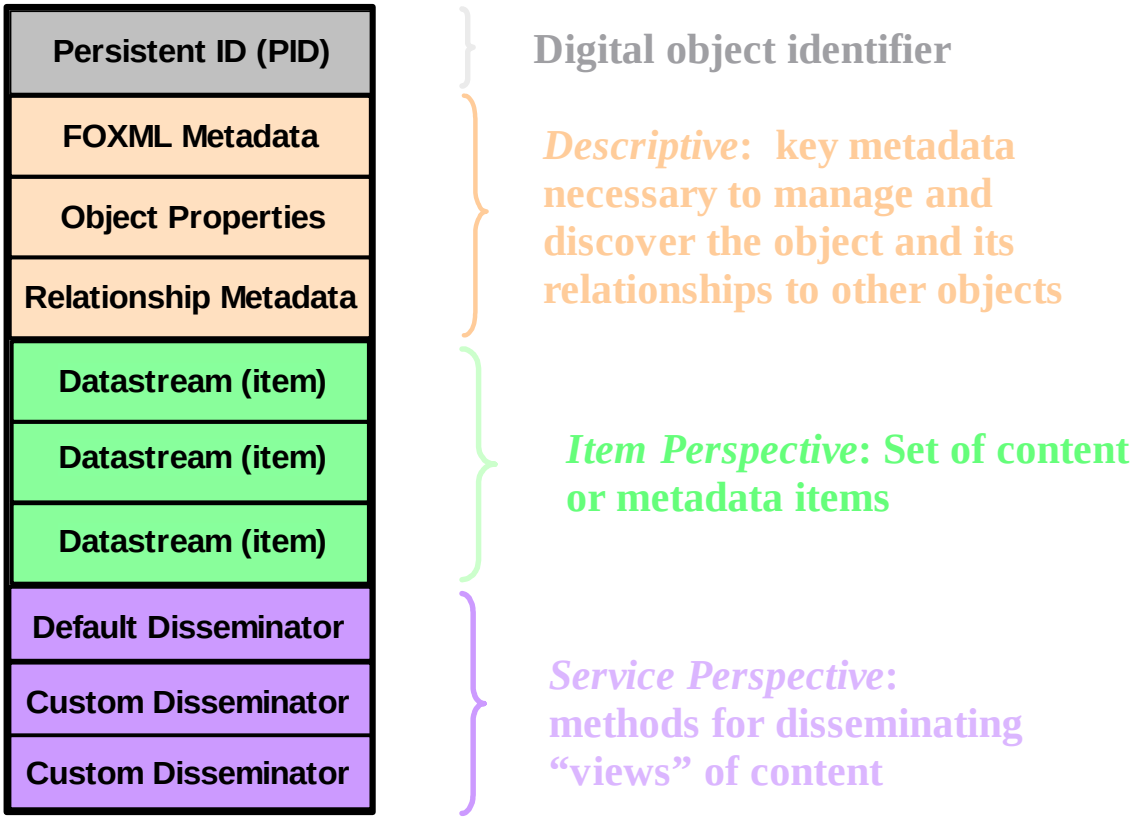

Figure 1: Fedora Digital Object Architectural View

A Fedora digital object consists of four parts

1. **Digital Object Identifier:** A unique, persistent identifier for the digital object.
2. **Descriptive Perspective:** The FOXML metadata for a digital object is the metadata that must be recorded with every digital object to facilitate the management of that object. FOXML metadata is distinct from other metadata that is stored in the digital object as content. This type of metadata is the metadata that is *required* by the Fedora repository architecture. All other metadata (e.g., descriptive metadata, technical metadata) is considered optional from the repository standpoint, and is treated as a datastream in a digital object.

Object Properties describe the object's type, its state, the content model to which it subscribes, the created and last modified dates of the object, and its label.

Relationship Metadata describes any relationships that exist between digital objects in a Fedora repository.

3. **Item Perspective:** a datastream is the component of a digital object that represents digital *content*. In other words, datastreams represent the digital stuff that is the essence of the digital object (e.g., digital images, encoded texts, audio recordings). All forms of metadata, except system metadata, are also treated as content, and are therefore represented as datastreams in a digital object. All datastreams have the potential to be disseminated from a digital object. A datastream can reference any type of content, and that content can be stored either locally or remotely to the repository system.
4. **Service perspective:** A disseminator is the component in a digital object that is used to associate behaviors (i.e., services) with the object. All Fedora objects have a default disseminator added at creation so that they can be immediately retrieved from the repository. Implementers may add any number of custom disseminations to their data objects.

Datastreams

A datastream is a component of a Fedora digital object. It represents some MIME-typed stream of content. The datastream is a description of this content and a pointer to the content's location. The datastream is not, however, equivalent to a file on a file system because the datastream may encapsulates bytestream content internally in the case of XML content stored with the object, but all other content is a reference to content that exists external to the repository.

Four Classifications for Datastreams

A Fedora object's datastreams must be one of the following four types

- **Managed (M):** Fedora stores and manages the bytestream
- **External (E):** Fedora stores a reference (URL) to the content
- **Redirect (R):** Fedora stores a reference (URL) to the content, but will not attempt to mediate access to the content
- **XML (X):** Fedora stores a namespaced block of XML content with the digital object. Most often, this type of datastream is used to add additional metadata. Every Fedora object has one XML metadata datastream added at object creation containing the Dublin Core metadata required for initial indexing.

Disseminators: A Look Under the Hood

The diagram below shows an example digital object modeled on the UVa MrSid Image digital object content model.

Basic Definition: Disseminator
A disseminator is the component in a digital object that is used to associate behaviors (i.e., services) with the object.

So a disseminator is a set of service subscriptions between the data object and a pair of behavior objects. The data object defines requirements for presentation of the data referenced within it by explicitly referencing the behavior objects.

Default Disseminator

Each Fedora data object is created with a **default disseminator**. The default disseminator allows repository administrator to get information about the object. For example, get the object's profile, list items/get item, list methods, get OAI_DC

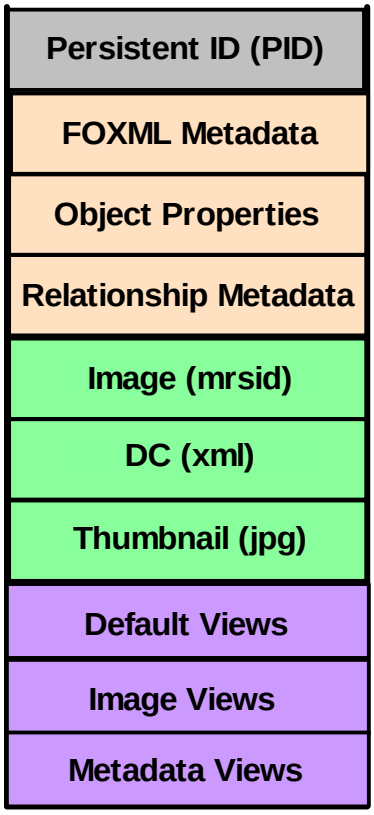

Figure 2: Fedora Digital Object Image Example

Custom Disseminator

Repository managers can optionally add as many custom disseminators to a Fedora object as they desire. In this way, each Fedora data object can be designed for maximum usability. In the example above, the **image views** disseminator allows users to retrieve the content of the object in the views designed by the repository administrators. In this example, a user could retrieve a thumbnail/preview sized image, a pre-defined medium-sized image, or a pre-defined high-resolution image. The latter of these are both generated from a MrSid encoded image, rather than retrieving a static version. The **metadata views** retrieve the metadata from the object. Users may retrieve the Dublic Core metadata or metadata from an XML type datastream, or both.

Section 4: Fedora Repository Server

Thus far, we have talked about the component parts of a Fedora repository, but the larger picture is also important. A repository is made up of digital objects, but in what context do those objects exist and how is it that users interact with them?

Fedora Server Architecture

This diagram shows in very general terms the structure of the entire repository. Users interact with the content of the repository by means of client applications, web browsers, batch programs, or server applications. These applications access the repository's data by means of the four APIs by which Fedora is exposed: management, access, search, which are exposed via HTTP or SOAP, and the OAI provider API, which is exposed via HTTP.

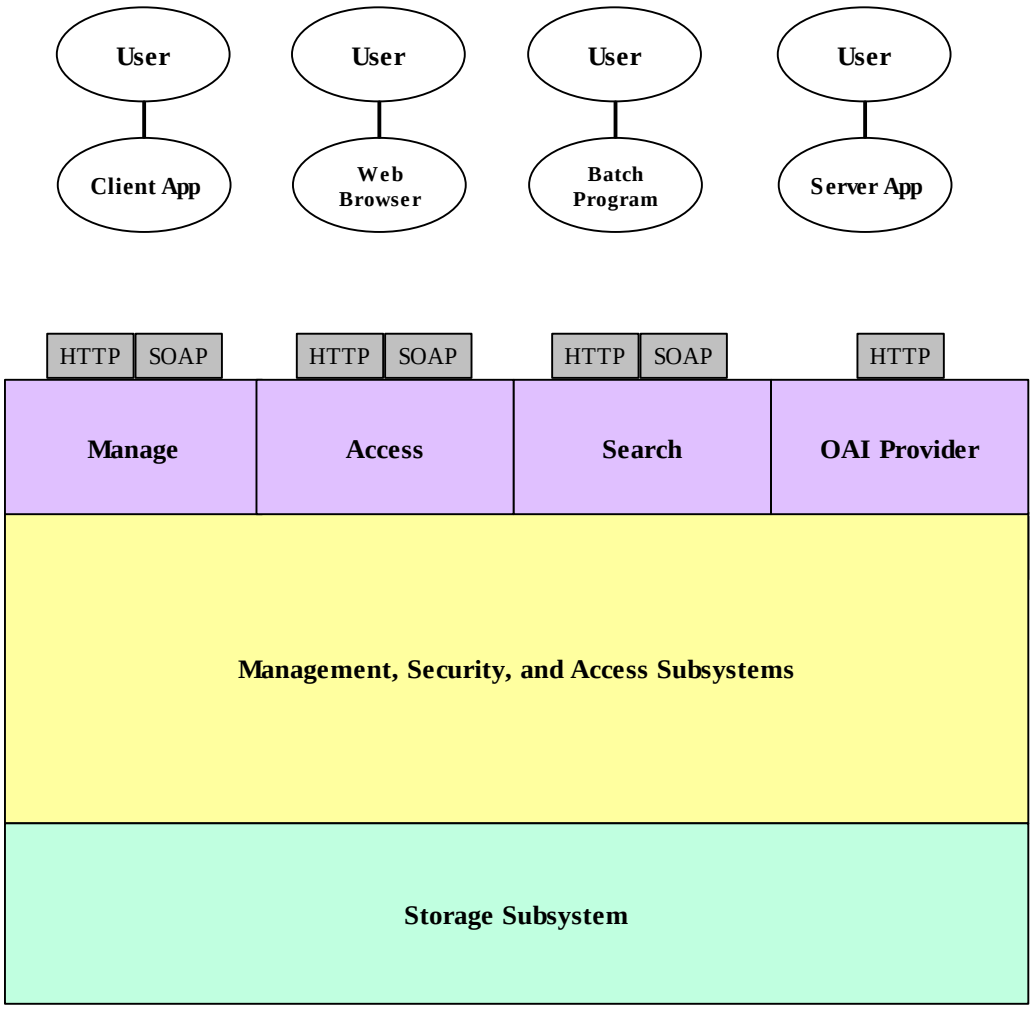

Figure 3: Fedora System Architecture (simplified)

Client and Web Service Interactions

This diagram gives another view of the larger context of a Fedora repository. Users perform common tasks such as ingesting objects, searching the repository, or accessing objects via client applications or a web browser. These client applications mediate this interaction with the repository via web services on the frontend, and on the backend, the repository interacts with web services to perform any data transformations that are requested by users. The transformed data is then passed back to the user via the frontend web services.

It is important to note that users only interact with the repository via the APIs, even though it may sometimes seem that they are interacting directly with an object, they are not.

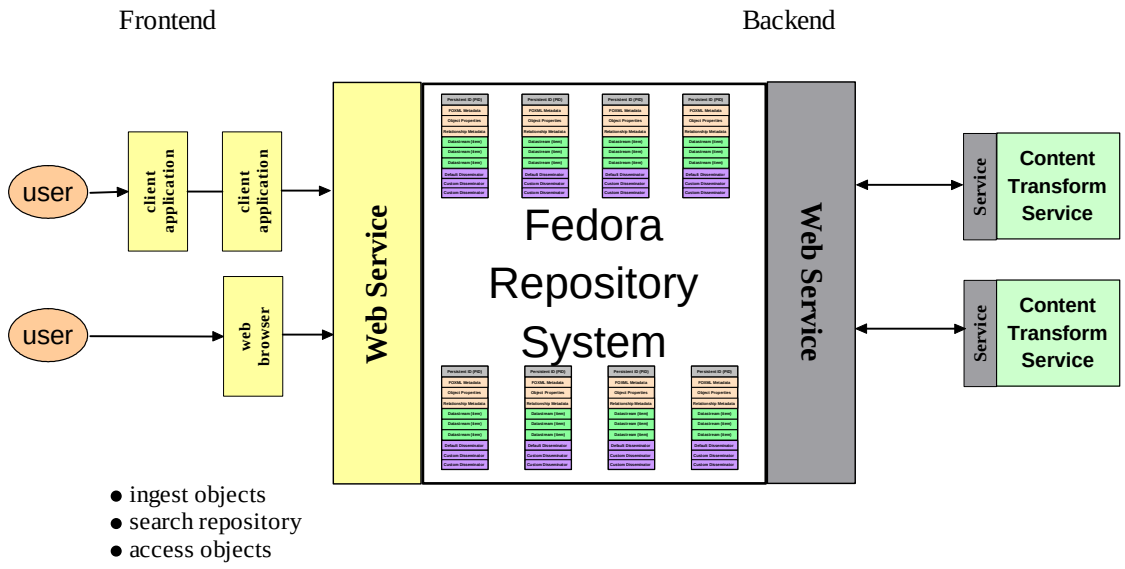

Figure 4: Client and Web Services Interaction